A reliable Turing machine

Ilir Çapuni   Peter Gács

May 18, 2017

Abstract

The title says it.

Contents

1 Introduction

1.1 To be written

1.2 Turing machines

Our contribution uses one of the standard definitions of a Turing machine.

A Turing machine M is defined by a tuple

$$(\Gamma, \Sigma, \tau, q_{\mathrm{start}}, F).$$

Here, Γ is a finite set of *states*, Σ is a finite alphabet, and

$$\tau \colon \Sigma \times \Gamma \to \Sigma \times \Gamma \times \{-1, 1\}$$

is the transition function. The tape alphabet Σ contains at least the distinguished symbols $\textvisiblespace, 0, 1$ where $\textvisiblespace$ is called the *blank symbol*. The state q_{start} is called the *starting state*, and there is a set F of *final states*.

The tape is blank at all but finitely many positions.

A *configuration* is a tuple

$$(q, A, h),$$

where $q \in \Gamma$ is the *current state*, $h \in \mathbb{Z}$ is the current *head position*, or *observed cell*, and $A \in \Sigma^{\mathbb{Z}}$ is the *tape content*: at position p, the tape contains the symbol $A(p)$. If $\xi = (q, A, h)$ is a configuration then we will write

$$\xi.\mathrm{state} = q, \quad \xi.\mathrm{tape} = A, \quad \xi.\mathrm{pos} = h. \tag{1.1}$$

Here, A is also called the *tape configuration*. The cell at position h is the *current cell*. Though the tape alphabet may contain non-binary symbols, we will restrict input and output to binary.

The transition function τ tells us how to compute the next configuration from the present one. When the machine is in a state q, at tape position h, and observes tape cell with content a, then denoting

$$(a', q', j) = \tau(a, q),$$

it will change the state to q', change the tape content at position h to a', and move to tape position to $h + j$. For $q \in F$ we have $a' = a$, $q' \in F$.

Definition 1.1 (Fault) We say that a *fault* occurs at time t if the output (a', q', j) of the transition function at this time is replaced with some other value (which is then used to compute the next configuration). ⌟

1.3 Codes, and the result

For fault-tolerant computation, some redundant coding of the information is needed.

Definition 1.2 (Codes) Let Σ_1, Σ_2 be two finite alphabets. A *block code* is given by a positive integer Q—called the *block size*—and a pair of functions

$$\psi_* : \Sigma_2 \to \Sigma_1^Q, \quad \psi^* : \Sigma_1^Q \to \Sigma_2$$

with the property $\psi^*(\psi_*(x)) = x$. It is extended to strings by encoding each letter individually: $\psi_*(x_1, \dots, x_n) = \psi_*(x_1) \cdots \psi_*(x_n)$. ⌟

For ease of spelling out a result, let us consider only computations whose outcome is a single symbol, at tape position 0.

Theorem 1 *There is a Turing machine M_1 with a function $a \mapsto a.output$ defined on its alphabet, such that for any Turing machine G with alphabet Σ and state set Γ there are $0 \le \varepsilon < 1$ and $\alpha_1, \alpha_2 > 0$ with the following property.*

For each input length $n = |x|$ a block code (φ_*, φ^*) of block size $Q = O((\log n)^{\alpha_1})$ can be constructed such that the following holds.

Let M_1 start its work from its initial state, and the initial tape configuration $\xi = (q_{\text{start}}, \varphi_*(x), 0)$. Assume further that during its operation, faults occur independently at random with probabilities $\le \varepsilon$.

Suppose that on input x machine G reaches a final state at time t and writes value y at position 0 of the tape. Then denoting by $\eta(u)$ the configuration machine M_1 at time u, at any time t' after

$$t \cdot (\log t)^{\alpha_2},$$

we have $\eta(t').\text{tape}(0).output = y$ with probability at least $1 - O(\varepsilon)$.

We emphasize that the actual code φ of the construction will depend on n only in a simple way: it will be the "concatenation" of one and the same fixed-size code with itself, $O(\log \log n)$ times.

2 Overview of the construction

A Turing machine that simulates "reliably" any other Turing machine even when it is subjected to isolated bursts of faults of constant size, is given in [1]. By *reliably* we mean that the simulated computation can be decoded from the history of the simulating machine despite occasional damages.

2.1 Isolated bursts of faults

Let us give a brief overview of a machine M_1 that can withstand isolated bursts of faults, as most of its construction will be reused in the probabilistic setting.

Let us break up the task of error correction into several problems to be solved. The solution of one problem gives rise to another problem one, but the process converges.

Redundant information The tape information of the simulated Turing machine will be stored in a redundant form, more precisely in the form of a block code.

Redundant processing The block code will be decoded, the retrieved information will be processed, and the result recorded. To carry out all this in a way that limits the propagation of faults, the tape will be split into tracks that can be handled separately, and the major processing steps will be carried out three times within one work period.

Local repair All the above process must be able to recover from a local burst of faults. For this, it is organized into a rigid, locally checkable structure with the help of local addresses, and some other tools like sweeps and short switchbacks (zigzags). The major tool of local correction, the local healing procedure, turns out to be the most complex part of the construction.

Disturbed local repair A careful organization of the healing procedure makes sure that even if a new burst interrupts it (or jumps into its middle), soon one or two new invocations of it will finish the job (whenever needed).

Here is some more detail. Each tape cell of the simulated machine M_2 will be represented by a block of size Q of the simulating machine M_1 called a *colony*. Each step of M_2 will be simulated by a computation of M_1 called a *work period*. During this time, the head of M_1 makes a number of sweeps over the current colony, decodes the represented cell symbol and state, computes the new state, and transfers the necessary information to the neighbor colony that corresponds to the new position of the head of M_2.

In order to protect information from the propagation of errors, the tape of M_1 is subdivided into *tracks*: each track corresponds to a *field* of a cell symbol of M_1 viewed as a data record. Each stage of computation will be repeated three times. The results will be stored in separate tracks, and a final cell-by-cell majority vote will recover the result of the work period from them.

All this organization is controlled by a few key fields, for example a field called *cAddr* showing the position of each cell in the colony, and a field *cSw* showing the last sweep of the computation (along with its direction) that has been performed already. The technically most challenging part of the construction is the protection of this control information from bursts.

For example, a burst can reverse the head in the middle of a sweep. Our goal is that such structural disruptions be discovered locally, so we cannot allow the head to go far from the place where it was turned back. Therefore the head's movement will not be straight even during a single sweep: it will make frequent zigzags. This will trigger the healing procedure if for example a turn-back is detected.

It is a significant challenge that the healing procedure itself can be interrupted (or started) by a burst.

Remark 2.1 This description uses some words in an informal way. Some of these will get precise definition later. For example, the word *colony* will have at least two formal definitions. From the point of view of the program (transition function), it is just a sequence of addresses from 0 to $Q-1$. From the point of view of the analysis of the behavior of the machine, it is a sequence of actual adjacent tape cells with the address field having theses values. ⌟

2.2 Hierarchical construction

In order to build a machine that can resist faults occurring independently of each other with some small probability, we take the approach suggested in [5], and implemented in [3] and [4] for the case of one-dimensional cellular automata, with some ideas from the tiling application of [2]. We will build a *hierarchy of simulations*: machine M_1 simulates machine M_2 which simulates machine M_3, and so on. For simplicity we assume all these machines have the same program, and all simulations have the same block size.

One cell of machine M_3 is simulated by one colony of machine M_2. Correspondingly, one cell of M_2 is simulated by one colony of machine M_1. So one cell of M_3 is simulated by Q^2 cells of M_1. Further, one step of machine M_3 is simulated by one work period of M_2 of, say, $O(Q^2)$ steps.

One step of M_2 is simulated by one work period of M_1, so one step of M_3 is simulated by $O(Q^4)$ steps of M_1.

Per construction, machine M_1 can withstand bursts of faults whose size is $\leq \beta$ for some constant parameter β, that are separated by some $O(Q^2)$ fault-free steps. Machines M_2, M_3, ... have the same program, so it would be natural to expect that machine M_1 can withstand also some *additional*, larger bursts of size $\leq \beta Q$ if those are separated by at least $O(Q^4)$ steps.

But a new obstacle arises. On the first level, damage caused by a big burst spans several colonies. The repair mechanism of machine M_1 outlined in Section 2.1 above is too local to recover from such extensive damage. This cannot be allowed, since then the whole hierarchy would stop working. So we add a new mechanism to M_1 that, more modestly, will just try to restore a large enough portion of the tape, so it can go on with the simulation of M_2, even if all original information was lost. For this, M_1 may need to rewrite an area as large as a few colonies.

This will enable the low-level healing procedure of machine M_2 to restore eventually a higher-level order.

All machines above M_1 in the hierarchy are "virtual": the only hardware in the construction is machine M_1. Moreover, they will not be ordinary Turing machines, but *generalized* ones, with some new features that are not needed on the lowest level but seem necessary in a simulated Turing machine: for example they allow a positive distance between neighboring tape cells.

A tricky issue is "forced self-simulation": while we are constructing machine M_1 we want to give it the feature that it will simulate a machine M_2 that works just like M_1. The "forced" feature means that this simulation should work without any written program (that could be corrupted).

This will be achieved by a construction similar to the proof of the Kleene's fixed-point theorem (also called recursion theorem). We first fix a (simple) programming language to express the transition function of a Turing machine. We write an interpreter for it in this same language (just as compilers for the C language are sometimes written in C). The program of the transition function of M_2 (essentially the same as that of M_1) in this language, is a string that will be "hard-wired" into the transition function of M_1, so that M_1, at the start of each work period, can write it on a working track of the base colony. Then the work period will interpret it, applying it to the data found there, resulting in the simulation of M_2.

In this way, an infinite sequence of simulations arises, in order to withstand larger and larger but sparser and sparser bursts of faults.

Since the M_1 uses the universal interpreter, which in turns simulates the same program, it is natural to ask how machine M_1 simulates a given Turing machine G that does the actual useful computation? For this task, we set aside a separate track on each machine M_i, on which some arbitrary other Turing machine can be simulated. The higher the level of the machine M_k that performs this "side-simulation", the higher the reliability. Thus, only the simulations $M_k \to M_{k+1}$ are forced, without program (that is a hard-wired program): the side simulations can rely on written programs, since the firm structure in the hierarchy $M_1, M_2, \dots$ will support them reliably.

2.3 From combinatorial to probabilistic noise

The construction we gave in the previous subsection was related to increasing bursts that are not frequent. In essence, that noise model is combinatorial. To deal with probabilistic noise combinatorially, we stratify the set of faulty times *Noise* as follows. For a series of parameters β_k, V_k, we first remove "isolated bursts" of type (β_1, V_1) of elements of this set. (The notion of "isolated bursts" of type (β, V) will be defined appropriately.) Then, we remove isolated bursts of type (β_2, V_2) from the remaining set, and so on. It will be shown that with the appropriate choice of parameters, with probability 1, eventually nothing is left over from the set *Noise*.

A composition of two reliable simulations is even more reliable. We will see that a sufficiently large hierarchy of such simulations resists probabilistic noise.

2.4 Difficulties

Let us spell-out some of the main problems that the paper deals with, and some general ways they will be solved or avoided. Some more specific problems will be pointed out later, along with their solution.

Non-aligned colonies A large burst of M_1 can modify the order of entire colonies or create new ones with gaps between them.

To overcome this problem conceptually, we introduce the notion of a *generalized Turing machine* allowing for non-adjacent cells. Each such machine has a parameter B called the *cell body size*. The cell body size of a Turing machine in Section 1.2 would still remain 1.

No structure What to do when the head is in a middle of an empty area where no structure exists? To ensure reliable passage across such areas, we will try to keep everything filled with cells, even if these are not part of the main computation.

Clean areas Noise can create areas over which the predictability of the simulated machine is limited. In these areas the (on this level) invisible structure of the underlying simulation may be destroyed. These areas should not simply be considered blank, since blankness implies predictable behavior. We could call these areas "dirty", but we prefer to use only a positive terminology, and talk about the complement, namely *clean* intervals. There is a danger in using the negative terminology, since the transition properties will allow to make only be from cleanness, not from "dirtiness". The following example shows that when the head comes out of a clean area, it can be "sucked in", and its state can change.

Extending cleanness The definition of the generalized Turing machine stipulates a certain "magical" extension of clean intervals, and also a magical appearance of a clean "hole" around the head whenever it passes a certain amount of cumulative time in a small interval. (Of course, this property needs to be implemented in simulation, which is one of the main burdens of the actual construction.) Once an area is cleaned, it will be repopulated with new cells. Their content is not important, what matters is the restoration of predictability.

Rebuilding If local repair fails, a special rule will be invoked that reorganizes a larger part of the tape (of the size of a few colonies instead of only a few cells). This is the mechanism enabling the "magical" restoration on the next level.

Example 2.2 (Uncleanness) Consider two levels of simulation as outlined in Section 2.2: machine M_1 simulates M_2 which simulates M_3. The tape of M_1 is subdivided into colonies of size Q_1. The tape content of the current colony of level 1 represents not only the content of the currently observed tape cell of machine M_2, but also its state.

A burst on level 1 has size $O(1)$, while a burst on level 2 has size $O(Q_1)$. Suppose that such a burst happens at some time, while the head was performing a simulation in colony $C(x) = x + [0, Q_1)$, and just intending to leave it on the *left* for a long time. Let a burst $\mathbf{b}_1$ of level 2 change the right neighbor colony $C(x + Q_1)$ and the state represented by it completely, into the last stage of a work period with the head on the verge of leaving it on the *right*. After that, let the burst deposit the head onto $C(x)$ again, letting it finish its simulation and continue on the *left*, for a very long time.

Much later, the head returns to $C(x)$, and would still not visit $C(x+Q_1)$ by design. But when it is at the right edge of $C(x)$, a burst $\mathbf{b}_2$ of level 1 (which can happen essentially in every work period of level 1) moves the head over to the left edge of $C(x+Q_1)$. Here it will be captured, simulating the cell x of M_2 from a cell and machine state represented by colony $C(x+Q_1)$ as set up by the big burst $\mathbf{b}_1$ earlier.

From the point of view of the simulated machine M_2 (which does not see bursts of level 1), the head came close to the unclean area created by burst $\mathbf{b}_1$, and then the state changed: the head started moving right, in a new state. ⌟

2.5 A shortcut solution

A fault-tolerant one-dimensional cellular automaton is constructed in [4]. If our Turing machine could just simulate such an automaton, it would become fault-tolerant. This can indeed almost be done provided that the size of the computation is known in advance. The cellular automaton can be made finite, and we could define a "kind of" Turing machine with a *circular tape* simulating it. But this solution requires input size-dependent hardware.

It seems difficult to define a fault-tolerant sweeping behavior on a regular Turing machine needed to simulate cellular automaton, without recreating an entire hierarchical construction—as we are doing here.

3 Notation

Most notational conventions given here are common; some other ones will also be useful.

Natural numbers and integers By $\mathbb{Z}$ we denote the set of integers.

$$\mathbb{Z}_{>0} = \{x : x \in \mathbb{Z},\ x > 0\},$$
$$\mathbb{Z}_{\geq 0} = \mathbb{N} = \{x : x \in \mathbb{Z},\ x \geq 0\}.$$

Intervals We use the standard notation for intervals:

$$[a,b] = \{x : a \le x \le b\}, \quad [a,b) = \{x : a \le x < b\},$$
$$(a,b] = \{x : a < x \le b\}, \quad (a,b) = \{x : a < x < b\}.$$

We will also write $[a,b)$ in place of $[a,b) \cap \mathbb{Z}$, whenever this leads to no confusion. Instead of $[x+a, x+b)$, sometimes we will write

$$x + [a,b)\,.$$

Ordered pairs Ordered pairs are also denoted by (a,b), but it will be clear from the context whether we are referring to an ordered pair or open interval.

Comparing the order of a number and an interval For a given number x and interval I, we write

$$x \ge I$$

if for every $y \in I$, $x \ge y$.

Distance The distance between two real numbers x and y is defined in a usual way:

$$d(x,y) = |x - y|.$$

The *distance of a point x from interval I* is

$$d(x,I) = \min_{y \in I} d(x,y).$$

Ball, neighborhood, ring, stripe A *ball of radius $r > 0$, centered at x* is

$$B(x,r) = \{y : d(x,y) \le r\}.$$

An *r-neighborhood of interval I* is

$$\{x : d(x,I) \le r\}.$$

An *r-ring* around interval I is

$$\{x : d(x,I) \le r \text{ and } x \notin I\}.$$

An *r-stripe to the right of interval I* is

$$\{x : d(x,I) \le r \text{ and } x \notin I \text{ and } x > I\}.$$

Logarithms Unless specified differently, the base of logarithms throughout this work is 2.

4 Specifying a Turing machine

Let us introduce the tools allowing to describe the reliable Turing machine.

4.1 Universal Turing machine

We will describe our construction in terms of universal Turing machines, operating on binary strings as inputs and outputs. We define universal Turing machines in a way that allows for rather general "programs".

Definition 4.1 (Standard pairing) For a (possibly empty) binary string $x = x(1)\cdots x(n)$ let us introduce the map

$$\langle x\rangle = 0^{|x|}1x,$$

Now we encode pairs, triples, and so on, of binary strings as follows:

$$\langle s,t\rangle = \langle s\rangle t,$$
$$\langle s,t,u\rangle = \langle\langle s,t\rangle,u\rangle,$$

and so on.

From now on, we will assume that our alphabets Σ, Γ are of the form $\Sigma = \{0,1\}^s$, $\Gamma = \{0,1\}^g$, that is our tape symbols and machine states are viewed as binary strings of a certain length. Also, if we write $\langle i,u\rangle$ where i is some number, it is understood that the number i is represented in a standard way by a binary string. ⌟

Definition 4.2 (Computation result, universal machine) Assume that a Turing machine M starting on binary x, at some time t arrives at the first time at some final state. Then we look at the longest (possibly empty) binary string to be found starting at position 0 on the tape, and call it the *computation result* $M(x)$. We will write

$$M(x,y) = M(\langle x,y\rangle),\quad M(x,y,z) = M(\langle x,y,z\rangle),$$

and so on.

A Turing machine U is called *universal* among Turing machines with binary inputs and outputs, if for every Turing machine M, there is a binary string p_M such that for all x we have $U(p_M,x) = M(x)$. (This equality also means that the computation denoted on the left-hand side reaches a final state if and only if the computation on the right-hand side does.) ⌟

Let us introduce a special kind of universal Turing machines, to be used in expressing the transition functions of other Turing machines. These are just the Turing machines for which the so-called s_{mn} theorem of recursion theory holds with $s(x, y) = \langle x, y\rangle$.

Definition 4.3 (Flexible universal Turing machine) A universal Turing machine will be called *flexible* if whenever p has the form $p = \langle p', p''\rangle$ then

$$U(p, x) = U(p', \langle p'', x\rangle).$$

Even if x has the form $x = \langle x', x''\rangle$, this definition chooses $U(p', \langle p'', x\rangle)$ over $U(\langle p, x'\rangle, x'')$, that is starts with parsing the first argument (this process converges, since x is shorter than $\langle x, y\rangle$). ⌟

It is easy to see that there are flexible universal Turing machines. On input $\langle p, x\rangle$, a flexible machine first checks whether its "program" p has the form $p = \langle p', p''\rangle$. If yes, then it applies p' to the pair $\langle p'', x\rangle$. (Otherwise it just applies p to x.)

Definition 4.4 (Transition program) Consider an arbitrary Turing machine M with state set Γ, alphabet Σ, and transition function τ. A binary string π will be called a *transition program* of M if whenever $\tau(a, q) = (a', q', j)$ we have

$$U(\pi, a, q) = \langle a', q', j\rangle.$$

We will also require that the computation induced by the program makes $O(|p| + |a| + |q|)$ left-right turns, over a length tape $O(|p| + |a| + |q|)$. ⌟

The transition program just provides a way to compute the (local) transition function of M by the universal machine, it does not organize the rest of the simulation.

Remark 4.5 In the construction of universal Turing machines provided by the textbooks (though not in the original one given by Turing), the program is generally a string encoding a table for the transition function τ of the simulated machine M. Other types of program are imaginable: some simple transition functions can have much simpler programs. However, our fixed machine is good enough (similarly to the optimal machine for Kolmogorov complexity). If some machine U' simulates M via a very simple program q, then

$$M(x) = U'(q, x) = U(p_{U'}, \langle q, x\rangle) = U(\langle p_{U'}, q\rangle, x),$$

so U simulates this computation via the program $\langle p_{U'}, q\rangle$. ⌟

4.2 Rule language

In what follows we will describe the generalized Turing machines M_k for all k. They are all similar, differing only in the parameter k; the most important activity of M_k is to simulate M_{k+1}. The description will be uniform, except for the parameter k. We will denote therefore M_k simply by M, and M_{k+1} by M^*. Similarly we will denote the block size Q_k of the block code of the simulation simply by Q.

Instead of writing a huge table describing the transition function $\tau_k = \tau$, we present the transition function as a set of *rules*. It will be then possible to write one *interpreter* program that carries out these rules; that program can be written for some fixed flexible universal machine Univ.

Each rule consists of some (nested) conditional statements, similar to the ones seen in an ordinary program: "**if** *condition* **then** *instruction* **else** *instruction*", where the condition is testing values of some fields of the state and the observed cell, and the instruction can either be elementary, or itself a conditional statement. The elementary instructions are an *assignment* of a value to a field of the state or cell symbol, or a command to move the head. Rules can call other rules, but these calls will never form a cycle. Calling other rules is just a shorthand for nested conditions.

Even though rules are written like procedures of a program, they describe a single transition. When several consecutive statements are given, then they change different fields of the state or cell symbol, so they can be executed simultaneously.

Assignment of value x to a field y of the state or cell symbol will be denoted by $y \leftarrow x$. We will also use some conventions introduced by the C language: namely, $x \leftarrow x + 1$ and $x \leftarrow x - 1$ are abbreviated to $x{+}{+}$ and $x{-}{-}$ respectively.

Rules can also have parameters, like `Swing`(a, b, u, v). Since each rule is called only a constant number of times in the whole program, the parametrized rule can be simply seen as a shorthand.

Mostly we will describe the rules using plain English, but it should always be clear that they are translatable into such rules.

For the machine M we are constructing, each state will be a tuple $q = (q_1, q_2, \dots, q_k)$, where the individual elements of the tuple will be called *fields*, and will have symbolic names. For example, we will have fields *Addr* and *Drift*, and may write q_1 as $q.Addr$ or just *Addr*, q_2 as $q.Drift$ or *Drift*, and so on.

Similarly for tape symbols. In order to distinguish fields of tape symbols from fields of the state, we will always start the name of a field of the tape symbols by the letter *c*. We have seen already one example of this, the field *cDir* of tape symbols in the definition of a generalized Turing machine.

In what follows we describe some of the most important fields we will use; others will be introduced later.

A properly formatted configuration of M splits the tape into blocks of Q consecutive cells called *colonies*. One colony of the tape of the simulating machine represents one cell of the simulated machine. The colony that corresponds to the cell that the simulated machine is scanning is called the *base colony* (a precise definition will be based on the actual history of the work of M). Once the the direction of the simulated head movement, called the *drift*, is known, the union of the base colony with the target colony in the direction of the drift (with a possible "bridge" betweend them) is called the *workspace* (this notion will need to be defined more carefully later).

There will be a field of the state called the *mode*:

$$\mathit{Mode} \in \{\mathrm{Normal}, \mathrm{Healing}, \mathrm{Rebuilding}\}.$$

In the *normal* mode, the machine is engaged in the regular business of simulation. The *healing* mode tries to correct some local fault due to a couple of neighboring bursts, while the *rebuilding* mode attempts to restore the colony structure on the scale of a couple of colonies.

The content of each cell of the tape of M also has several fields. Some of these have names identical to fields of the state. In describing the transition rule of M we will write, for example, $q.\mathit{Addr}$ simply as Addr, and for the corresponding field of the observed cell symbol a we will write $a.\mathit{cInfo}$, or just cInfo. The array of values of the same field of the cells will be called a *track*. Thus, we will talk about the *cHold* track of the tape, corresponding to the *cHold* field of cells.

Each field of a cell has also a possible value $\emptyset$ whose approximate meaning is "undefined".

Some fields and parameters are important enough to introduce them right away. The

$$\mathit{cInfo}, \mathit{cState}$$

track of a colony of M contain the strings that encode the content of the simulated cell of M^* and its simulated state respectively.

$$cProg$$

track stores the program of M^*, in an appropriate form to be interpreted by the simulation. The field

$$cAddr$$

of the cell shows the position of the cell in its colony: it takes values in $[-Q, 2Q)$, since the addresses in a bridge (see later) will be continuations of those in the colony (which run from 0 to $Q-1$). There is a corresponding *Addr* field of the state.

The direction in $\{-1, 1\}$ in which the simulated head moves will be denoted by

$$Drift.$$

There is a corresponding field *cDrift*. The number of the last sweep of the work period will depend on the drift d, and will be denoted by

$$\mathrm{Last}(d). \tag{4.1}$$

The colony along with the adjacent cells that continue its addresses will be called an *extended colony*. A colony can only be extended in the direction of the drift. The

$$Sw$$

field counts the sweeps that the head makes during the work period. There is a corresponding *cSw* field in the cell. In calculating parameters, we will make use of

$$V = \max(\mathrm{Last}(-1), \mathrm{Last}(1)). \tag{4.2}$$

Cells will be designated as belonging to a number of possible *kinds*, signaled by the field

$$cKind$$

with values

$$\text{Member}, \text{Target}, \mathit{Vac}, \text{Stem}.$$

Here is a description of the role of these cell kinds. Normally, cells will have the kind Member. During the simulation, however, the elements of the colony that is to become the next base colony, will be made to have the kind Target. If the neighbor colony is not adjacent, then the base colony will be extended to it by a *bridge* of up to $Q-1$ adjacent cells. Bridge cells are member cells with addresses outside $[0, Q)$. Though they have the kind Member, they will frequently be treated differently from the other member cells.

The stem kind is sometimes convenient when some cells need to be created temporarily that do not participate in any known colony structure. We will also try to keep all areas between colonies filled with (not necessarily adjacent) stem cells. For example the computation may find that a colony does not properly encode a tape cell by the required error-correcting code. Then we want to "kill" the whole colony. This will happen by turning the kind of each of its cell to Stem.

During healing, some special fields of the state and cell are used, they will be subfields of the field

$$\mathit{Heal}. \tag{4.3}$$

In particular, there will be a *Heal.Sw* field.

Healing only changes the cells that need to be changed. But during rebuilding, the tape will also be used. we will work with subfields of the field *Rebuild*, and a cell will be called *marked for rebuilding* if $\mathit{Rebuild.cSw} \neq 0$.

5 Exploiting structure in the noise

5.1 Sparsity

Let us introduce a technique connecting the combinatorial and probabilistic noise models.

Definition 5.1 (Centered rectangles, isolation) Let $\mathbf{r} = (r_1, r_2)$, $r_1, r_2 \geq 0$, be a two-dimensional nonnegative vector. An *rectangle* of radius $\mathbf{r}$ *centered* at $\mathbf{x}$ is

$$B(\mathbf{x}, \mathbf{r}) = \{\mathbf{y} : |y_i - x_i| \leq r_i, i = 1, 2\}. \tag{5.1}$$

Let $E \subseteq \mathbb{Z}^2$ be a two-dimensional set. A point $\mathbf{x}$ of E is $(\mathbf{r}, \mathbf{r}^*)$-*isolated* if

$$E \cap B(\mathbf{x}, \mathbf{r}^*) \subseteq B(\mathbf{x}, \mathbf{r}).$$

Set E is $(\mathbf{r}, \mathbf{r}^*)$-*sparse* if $D(E, \mathbf{r}, \mathbf{r}^*) = \emptyset$, that is it consists of $(\mathbf{r}, \mathbf{r}^*)$-isolated points. Let

$$D(E, \mathbf{r}, \mathbf{r}^*) = \{\mathbf{x} \in E : \mathbf{x} \text{ is not } (\mathbf{r}, \mathbf{r}^*)\text{-isolated from } E\}. \tag{5.2}$$

$\lrcorner$

Definition 5.2 (Sparsity) Let

$$\beta \geq 9, \gamma \gg \beta \tag{5.3}$$

be constants (we will choose γ sufficiently large as the proof requires), and let $0 < B_1 < B_2 < \cdots$, $T_1 < T_2 < \cdots$, be sequences of positive integers to be fixed later.

For a two-dimensional set E, let $E^{(1)} = E$. For $k > 1$ we define recursively:

$$E^{(k+1)} = D(E^{(k)}, \beta(B_k, T_k), \gamma(B_{k+1}, T_{k+1})). \tag{5.4}$$

Set $E^{(k)}$ is called the *k-th residue* of E. It is *k-sparse* if $E^{(k+1)} = \emptyset$. It is simply *sparse* if $\bigcap_k E^{(k)} = \emptyset$.

When $E = E^{(k)}$ and k is known then we will denote $E^{(k+1)}$ simply by E^*. $\lrcorner$

The following lemma connects the above defined sparsity notions to the requirement of small fault probability. It is formulated somewhat redundantly, for easier application.

Lemma 5.3 (Sparsity) *Let $Q_k = B_{k+1}/B_k$, $U_k = T_{k+1}/T_k$, and*

$$\lim_{k \to \infty} \frac{\log(U_k Q_k)}{1.5^k} = 0. \tag{5.5}$$

For sufficiently small ε, for every $k \geq 1$ the following holds. Let $E \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ be a random set with the property that each pair (p, t) belongs to E independently from the other ones with probability $\leq \varepsilon$.

Then for each point $\mathbf{x}$ and each k,

$$\mathsf{P}\{B(\mathbf{x}, (B_k, T_k)) \cap E^{(k)} \neq \emptyset\} < 2\varepsilon \cdot 2^{-1.5^k}.$$

As a consequence, the set E is sparse with probability 1.

Proof. Let $k = 1$. Rectangle $B(\mathbf{x}, (B_1, T_1))$ is a single point, hence the probability of our event is $< \varepsilon$. Let us prove the inequality by induction, for $k + 1$.

Note that our event depends at most on the rectangle $B(\mathbf{x}, 3(B_k, T_k))$. Let

$$p_k = 2\varepsilon \cdot 2^{-1.5^k}.$$

Suppose $\mathbf{y} \in E^{(k)} \cap B(\mathbf{x}, \gamma(B_{k+1}, T_{k+1}))$. Then, according to the definition of $E^{(k)}$, there is a point

$$\mathbf{z} \in B(\mathbf{y}, \gamma(B_{k+1}, T_{k+1})) \cap E^{(k)} \setminus B(\mathbf{y}, \beta(B_k, T_k)). \tag{5.6}$$

Consider a standard partition of the (two-dimensional) space-time into rectangles $K_p = \mathbf{c}_p + [-B_k, B_k) \times [-T_k, T_k)$ with centers $\mathbf{c}_1, \mathbf{c}_2, \ldots$. The rectangles K_i, K_j containing $\mathbf{y}$ and $\mathbf{z}$ respectively intersect $B(\mathbf{x}, 2\gamma(B_{k+1}, T_{k+1}))$. The triple-size rectangles $K'_i = c_i + [-3B_k, 3B_k) \times [-3T_k, 3T_k)$ and K'_j are disjoint, since (5.3) and (5.6) imply $|y_1 - z_1| > \beta B_k$ and $|y_2 - z_2| > \beta T_k$.

The set $E^{(k)}$ must intersect two rectangles K_i, K_j of size $2(B_k, T_k)$ separated by at least $4(B_k, T_k)$, of the big rectangle $B(\mathbf{x}, 2\gamma(B_{k+1}, T_{k+1}))$.

By the inductive hypothesis, the event $\mathcal{F}_i$ that K_i intersects E_k has probability bound p_k. It is independent of the event $\mathcal{F}_j$, since these events depend only on the triple size disjoint rectangles K'_i and K'_j.

The probability that both of these events hold is at most p_k^2. The number of possible rectangles K_p intersecting $B(\mathbf{x}, 2\gamma(B_{k+1}, T_{k+1}))$ is at most $C_k := ((2\gamma^2 U_k Q_k) + 2)^2$, so the number of possible pairs of rectangles is at most $C_k^2/2$, bounding the probability of our event by

$$\begin{aligned} C_k^2 p_k^2/2 &= 2C_k^2 \varepsilon^2 2^{-1.5^{k+1}} \cdot 2^{-0.5 \cdot 1.5^k} \\ &= 2\varepsilon 2^{-1.5^{k+1}} \cdot \varepsilon C_k^2 2^{-0.5 \cdot 1.5k}. \end{aligned}$$

Since $\lim_k \frac{\log(U_k Q_k)}{1.5^k} = 0$, the last factor is ≤ 1 for sufficiently small ε. □

5.2 Error-correcting code

Let us add error-correcting features to block codes introduced in Definition 1.2.

Definition 5.4 (Error-correcting code) A block code is (β, t)-*burst-error-correcting*, if for all $x \in \Sigma_2$, $y \in \Sigma_1^Q$ we have $\psi^*(y) = x$ whenever y differs from $\psi_*(x)$ in at most t intervals of size $\le \beta$.

For such a code, we will call a word $y \in \Sigma_1^Q$ is r-*compliant* if it differs from a codeword of the code by at most r intervals of size $\le \beta$. ⌟

Example 5.5 (Repetition code) Suppose that $Q \ge 3\beta$ is divisible by 3, $\Sigma_2 = \Sigma_1^{Q/3}$, $\psi_*(x) = xxx$. Let $\psi^*(y)$ be obtained as follows. If $y = y(1) \dots y(Q)$, then $x = \psi^*(y)$ is defined as follows: $x(i) = \mathrm{maj}(y(i), y(i+Q/3), y+2Q/3)$. For all $\beta \le Q/3$, this is a $(\beta, 1)$-burst-error-correcting code.

If we repeat 5 times instead of 3, we get a $(\beta, 2)$-burst-error-correcting code. Let us note that there are much more efficient such codes than just repetition. ⌟

Consider a Turing machine $(\Gamma, \Sigma, \tau, q_{\text{start}}, F)$ (actually a generalized one to be defined later) simulating some Turing machine $(\Gamma^*, \Sigma^*, \tau^*, q^*_{\text{start}}, F^*)$. We will assume that $\Gamma^* \cup \{\emptyset\}$, and the alphabet Σ^* are subsets of the set of binary strings $\{0,1\}^l$ for some $l < Q$ (we can always ignore some states or tape symbols, if we want).

Definition 5.6 (Interior) Consider an interval I of neighbor cells. A cell belongs to the *interior* of I if there are at least *PadLen* neighbors between it and the complement of I. In particular, we will talk about the interior of a colony and the interior of a workspace. ⌟

We will store the coded information in the interior of the colony, since it is more exposed to errors near the boundaries. So let (υ_*, υ^*) be a $(\beta, 2)$-burst-error-correcting block code

$$\upsilon_* : \{0,1\}^l \cup \{\emptyset\} \to \{0,1\}^{(Q - 2\cdot \mathit{PadLen})B}.$$

We could use, for example, the repetition code of Example 5.5. Other codes are also appropriate, but we require that they have some fixed programs p_{encode}, p_{decode} on the universal machine Univ, in the following sense:

$$\upsilon_*(x) = \mathrm{Univ}(p_{\text{encode}}, x), \quad \upsilon^*(y) = \mathrm{Univ}(p_{\text{decode}}, y).$$

Also, these programs must work in quadratic time and linear space on a one-tape Turing machine (as the repetition code certainly does).

Let us now define the block code (ψ_*, ψ^*) used in the definition of the configuration code (φ_*, φ^*) as outlined in Section 6.3. We define

$$\psi_*(a) = 0^{\mathit{PadLen}} \upsilon_*(a) 0^{\mathit{PadLen}}. \tag{5.7}$$

It will be easy to compute the configuration code from ψ_*, once we know what fields there are which ones need initialization.

The decoded value $\psi^*(x)$ is obtained by first removing *PadLen* symbols from both ends of x to get x', and then computing $v^*(x')$.

6 The model

Recall the definition of sparsity in Section 5.1: there will be a sequence $0 < B_1 < B_2 < \cdots$ of "scales" in space and a sequence $T_1 < T_2 < \cdots$ of scales in time, and a constant β. We will define a sequence of simulations $M_1 \to M_2 \to \cdots$ where each M_k is a machine simulating one on a higher level. For simplicity, we will use the notation $M = M_k$, $M^* = M_{k+1}$, and similarly for the other parameters, for example B, T, Q, U. As already indicated in Section 2.2, these machines will generalize ordinary Turing machines, with a number of new features.

6.1 Generalized Turing machine

Standard Turing machines do not have operations like "creation" or "killing" of cells, nor do they allow for cells to be non-adjacent. We introduce here a *generalized Turing machine*. It depends on an integer $B \geq 1$ that denotes the cell body size, and an upper bound T on the transition time, as well as a *pass number* π, whose meaning will be explained in the definition of configurations. These parameters are convenient since they provide the illusion that the different Turing machines in the hierarchy of simulations all operate on the same linear space. Even if the notions of cells, alphabet and state are different for each machine of the hierarchy, at least the notion of a *location on the tape* is the same.

Definition 6.1 (Generalized Turing machine) A *generalized Turing machine* M is defined by a tuple

$$(\Gamma, \Sigma, \tau, \mathit{NonAdj}, \mathit{cDir}, q_{\text{start}}, F, B, T, \pi), \tag{6.1}$$

where Γ and Σ are finite sets called the *set of states* and the *alphabet* respectively,

$$\tau : \Sigma \times \Gamma \to \Sigma \times \Gamma \times \{-1, 1\},$$

is the *transition function*. *NonAdj* is a function of the state (can be called a "field" if a state is viewed as a piece of data, a record with several fields). $\mathit{NonAdj}\colon \Gamma \to \{\mathsf{true}, \mathsf{false}\}$ will show whether the last move was from a non-adjacent cell. The function $\mathit{cDir}\colon \Sigma \to \{-1, 1\}$ of the cell content needs to always point towards the head, so the transition function τ is required to have the property that if $(a', q', j) = \tau(a, q)$ then $a'.\mathit{cDir} = j$.

The role of starting state q_{start} and final states in F is as before. The integer $B \geq 1$ is called the *cell body size*, and the real number T is a bound on the transition time. The *pass number* π will play a role in the definition of trajectories.

Among the elements of the tape alphabet Σ, we distinguish the elements $0, 1, \mathit{Bad}, \mathit{Vac}$. The role of the symbols *Bad* and *Vac* will be clarified below.

$\lrcorner$

Definition 6.2 (Configuration) Consider a generalized Turing machine (6.1). A *configuration* is a tuple

$$(q, A, h, \hat{h},),$$

where $q \in \Gamma$, $A : \mathbb{Z} \to \Sigma$, $h, \hat{h} \in \mathbb{Z}$. The array A is the tape configuration. As in (1.1) before, h is the head position, but it may differ from the *current cell* $\hat{h}$, so now if $\xi = (q, A, h, \hat{h},)$ then we write

$$\xi.\text{state} = q, \quad \xi.\text{tape} = A, \quad \xi.\text{pos} = h, \quad \xi.\text{cur-cell} = \hat{h}. \qquad (6.2)$$

A point p is *clean* if $A(p) \neq \mathit{Bad}$. A set of points is *clean* if it consists of clean points.

We say that there is a *cell* at a position $p \in \mathbb{Z}$ if the interval $p + [0, B)$ is clean and $A(p) \neq \mathit{Vac}$. In this case, we call the interval $p + [0, B)$ the *body* of this cell. Cells must be at distance $\geq B$ from each other, that is their bodies must not intersect. They are called *adjacent* if the distance is exactly B. A configuration will always have the following property:

(C1) If a maximal clean interval has an endpoint to the right (left) then it ends in a cell body in that direction.

For all cells p, the value $A(p).\mathit{cDir}$ is required to point towards the head position h, that is

$$A(p).\mathit{cDir} = \operatorname{sign}(h - p).$$

Whenever the interval $h + [3B, 3B)$ is clean there must be a cell at some position $\hat{h}$ within this interval called the *current cell*, with a body within $2B$ from h.

The array A is *Vac* everywhere but in finitely many positions. Let

$$\mathrm{Configs}_M$$

denote the set of all possible configurations of a Turing machine M. ⌟

All the above definitions can clearly be localized to define a configuration *over a space interval* I, where it is always understood that $h \in I$, that is I contains the head.

Definition 6.3 (Local configuration, replacement) A *local configuration on* a (finite or infinite) interval I is given by values assigned to the cells of I, along with the following information: whether the head is to the left of, to the right of or inside I, and if it is inside, on which cell, and what is the state.

If I' is a subinterval of I, then a local configuration ξ on I clearly gives rise to a local configuration $\xi(I')$ on I' as well, called its *subconfiguration*: If the head of ξ was in I and it was for example to the left of I', then now $\xi(I')$ just says that it is to the left, without specifying position and state.

Let ξ be a configuration and $\zeta(I)$ a local configuration that contains the head if and only if $\xi(I)$ contains the head. Then the configuration $\xi|\zeta(I)$ is obtained by replacing ξ with ζ over the interval I, further if ξ contains the head then also replacing ξ.pos with ζ.pos and ξ.state with ζ.state. ⌟

It is natural to name a sequence of configurations that is conceivable as a computation (faulty or not) of a Turing machine as "history". The histories that obey the transition function then could be called "trajectories". In what follows we will stretch this notion to encompass also some limited violations of the transition function.

In connection with any underlying Turing machine with a given starting configuration, we will denote by

$$\mathit{Noise} \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0} \tag{6.3}$$

the set of space-time points (p, t), such that a fault occurs at time t when the head is at position p.

Definition 6.4 (History) Let us be given a generalized Turing machine (6.1). Consider a sequence $\eta = (\eta(0), \eta(1), \dots)$ of configurations with $\eta(t) =$

$(q(t), A(\cdot, t), h(t), \hat{h}(t))$, along with a noise set *Noise*. The *switching times* of this sequence are the times t when one of the following changes: the state, the content or position of the current cell. In other words, if the triple $(q(t), A(\hat{h}(t), t), \hat{h}(t))$ changes. The interval between two consecutive switching times is the *dwell period.* The pair

$$(\eta, \mathit{Noise})$$

will be called a *history* of machine M if the following conditions hold.

- We have $|h(t) - h(t')| \leq |t' - t|$.
- In two consecutive configurations, content $A(p, t)$ of the positions p not in $h(t) + [-B, B)$, remains the same: for example $A(n, t+1) = A(n, t)$ for all $n \notin h(t) + [-B, B)$.
- At each noise-free switching time the head is on the new current cell, that is $\hat{h}(t) = h(t)$. In particular, when at a switching time a current cell becomes *Vac*, the head must already be on another (current) cell.
- The length of any noise-free dwell period in which the head is staying on clean positions is at most T.

Let

$$\mathrm{Histories}_M$$

denote the set of all possible histories of M.

We say that a cell *dies* in a history if it becomes *Vac*.

It is clear that all the above definition can be *localized* to define a history *over a space-time rectangle* $I \times J$, where it is always understood that $h \in I$ for all times $t \in J$, that is I contains the head throughout the time interval considered.

⌟

The transition function τ of a generalized Turing machine imposes constraints on histories: those histories obeying the constraints will be called trajectories.

Definition 6.5 Suppose that at times t' before a switching time t but after any previous switch, the machine is in a state $q = q(t')$, with current cell $x = \hat{h}(t')$, with cell content $a = A(\hat{h}(t'), \hat{h}(t'))$. We say that the new state, the new content of cell x (including when it dies), and the direction of the new position y from x are *dictated by the transition function* if

$$(q(t), A(\hat{h}(t), \hat{h}(t)), \operatorname{sign}(\hat{h}(t) - \hat{h}(t'))) = \tau(a, q),$$

further $q(t).\mathit{NonAdj} = \mathsf{true}$ if and only if $\hat{h}(t) - \hat{h}(t') \notin \{-B, 0, B\}$. In other words, if $(q', a', j) = \tau(q, a)$ then the new state is q', the new content of x is a', the direction of y from x is j, and $q'.\mathit{NonAdj} = \mathsf{true}$ only if y is not equal or adjacent to x. ⌟

The following definitions will use the constants

$$c_{\text{attack}} = 6, c_{\text{spill}} = 5, c_{\text{dwell}}. \tag{6.4}$$

Definition 6.6 If in a configuration the head is in a clean interval of size $\geq 1.5c_{\text{attack}}B$, at a distance $\geq B$ from the complement, then we will say that the head is *inside a clean hole.* ⌟

Definition 6.7 (Trajectory) A history (η, Noise) of a generalized Turing machine (6.1) with $\eta(t) = (q(t), A(t), h(t), \hat{h}(t))$ is called a *trajectory* of M if the following conditions hold, in any noise-free time interval J.

Transition Function Suppose that there is a switch, and the current cell x is inside the clean area, by a distance of at least $2.5B$ in the direction of the new cell y, and by at least $0.5B$ in the other direction. Then the new state, the cell content of x (including when it dies), and the direction of y from x are dictated by the transition function. If y did not exist before then it is adjacent to x. Nothing else changes on the tape.

Further the length of the dwell period is bounded by T.

Spill Bound A clean space interval may shrink by at most $c_{\text{spill}}B$ on either side.

Dwell Cleaning If the head spends time $c_{\text{dwell}}T$ *cumulatively* during J in any interval I of size $1.5c_{\text{attack}}B$ (counting the times spent in I and not the times spent outside I) then at some time during this, it will be inside a clean hole intersecting I.

Attack cleaning Suppose that the current cell x, is the right endcell of a maximal clean interval of size $\geq (c_{\text{attack}} + 1)B$. Suppose further that the transition function directs the head right. Then by the time the head comes back to $x - c_{\text{attack}}B$, the right end of the clean interval containing it advances to the right by at least B.

A similar property is required when "left" and "right" are interchanged.

Pass Cleaning If the head passes π times an interval $[a, b)$ then the subinterval $[a + c_{\mathrm{spill}}B, b - c_{\mathrm{spill}}B)$ becomes clean.

⌟

The Spill Bound property makes the following notation convenient:

$$\beta' = \beta + 2c_{\mathrm{spill}}. \tag{6.5}$$

The above definition can also clearly be localized to some space-time rectangle just as the definition of history was.

The Attack Cleaning property says essentially that if the head moves out on the right end of a clean interval then next time it comes in, it must extend the right end of the interval by at least B (while temporarily possibly withdrawing it by $c_{\mathrm{spill}}B$).

6.2 Simulation

Until this moment, we used the term "simulation" informally, to denote a correspondence between configurations of two machines which remains preserved during the computation. In the formal definition, this correspondence will essentially be a code $\varphi = (\varphi_*, \varphi^*)$. The *decoding* part of the code is the more important. We want to say that machine M_1 simulates machine M_2 via simulation φ if whenever (η, Noise) is a trajectory of M_1 then $(\eta^*, \mathit{Noise}^*)$, defined by $\eta^*(\cdot, t) = \varphi^*(\eta(\cdot, t))$, is a trajectory of M_1. Here, Noise^* is computed by an appropriate mapping.

We will make, however, two refinements. First, we may weaken the condition by requiring this only for those η for which the initial configuration $\eta(\cdot, 0)$ has been obtained by encoding, that is it has the form $\eta(\cdot, 0) = \varphi_*(\xi)$. The encoding function gets a role this way in the definition, after all.

But there is a more complex refinement. When a colony is in transition between encoding one simulated value to encoding another one, there may be times when the value represented by it before the transition is already not decodable from it, and the value after the transition is not yet decodable from it. So we will define simulation decoding as a mapping Φ^* between *histories*, not just configurations. This allows a certain amount of "looking back": the map Φ^* can depend on the configurations at the beginning of the "work period".

It is the mapping Φ^* that will also define Noise^*. A history was defined above in Definition 6.4 as a pair (η, Noise), so we will have $\Phi^*(\eta, \mathit{Noise}) =$

$(\eta^*, \mathit{Noise}^*)$. The meaning of Noise^* will be, just as in Definition 5.2 of sparsity: it will be obtained from Noise by deleting those small isolated parts that the error-correcting simulation can deal with.

Definition 6.8 (Simulation) Let M_1, M_2 be two generalized Turing machines, and let

$$\varphi_* : \mathrm{Configs}_{M_2} \to \mathrm{Configs}_{M_1}$$

be a mapping from configurations of M_2 to those of M_1, such that it maps starting configurations into starting configurations. We will call such a map a *configuration encoding*. Let

$$\Phi^* : \mathrm{Histories}_{M_1} \to \mathrm{Histories}_{M_2}$$

be a mapping. The pair (φ_*, Φ^*) is called a *simulation* (of M_2 by M_1) if for every trajectory (η, Noise) with initial configuration $\eta(\cdot, 0) = \varphi_*(\xi)$, the history $(\eta^*, \mathit{Noise}^*) = \Phi^*(\eta, \mathit{Noise})$ is a trajectory of machine M_2.

We say that M_1 *simulates* M_2 if there is a simulation (φ_*, Φ^*) of M_2 by M_1. ⌟

6.3 Hierarchical codes

Recall the notion of a code in Definition 1.2.

Definition 6.9 (Code on configurations) Consider two generalized Turing machines M_1, M_2 with the corresponding state spaces, alphabets and transition functions, and an integer $Q \geq 1$. We require

$$B_2 = QB_1. \tag{6.6}$$

Assume that a block code $\psi_* : \Sigma_2 \times (\Gamma_2 \cup \{\emptyset\}) \to \Sigma_1^Q$ is given, with an appropriate decoding function, ψ^*. With $(a, q) \in \Sigma_2 \times (\Gamma_2 \cup \{\emptyset\})$, symbol a is interpreted the content of some tape square. The value q is the state of M_2 provided the head is observing this square, and $\emptyset$ if it is not. For all a, q, if $\psi_*(a, q) = (b_1, \dots, b_Q)$ then we require $b_i.\mathit{cDir} = a.\mathit{cDir}$ for each i.

This block code gives rise to a *code on configurations*, that is a pair of functions

$$\varphi_* : \mathrm{Configs}_{M_2} \to \mathrm{Configs}_{M_1}, \quad \varphi^* : \mathrm{Configs}_{M_1} \to \mathrm{Configs}_{M_2}$$

that encodes some (initial) configurations ξ of M_2 into configurations of M_1. Let ξ be a configuration of M_2 with ξ.cur-cell $= \xi$.pos, ξ.state $= q_{\text{start}}$. We set $\varphi_*(\xi)$.pos $= \varphi_*$.cur-cell $= \xi$.pos, φ_*.state $= q^*_{\text{start}}$, the starting state of M_1, and

$$\varphi_*(\xi).\text{tape}(iB_2, \dots, (i+1)B_2 - B_1) = \psi_*(\xi.\text{tape}(i), s)$$

where $s = \xi$.state if $i = \xi$.pos, and $\emptyset$ otherwise. A configuration ξ is called a *code configuration* if it has the form $\xi = \varphi_*(\zeta)$. ⌟

Definition 6.10 (Hierarchical code) For $k \geq 1$, let Σ_k be an alphabet, Γ_k be a set of states of a generalized Turing machine M_k. Let $Q_k > 0$ be an integer colony size, let φ_k be a code on configurations defined by a block code

$$\psi_k : \Sigma_{k+1} \times (\Gamma_{k+1} \cup \{\emptyset\}) \to \Sigma_k^{Q_k}$$

as in Definition 6.9. The sequence of triples $(\Sigma_k, \Gamma_k, \varphi_k)$, $(k \geq 1)$, is called a *hierarchical code*. For the given hierarchical code, the configuration ξ^1 of M_1 is called a *hierarchical code configuration* if a sequence of configurations $\xi^2, \xi^3, \dots$ of $M_2, M_3, \dots$ exists with

$$\xi^k = \varphi_{*k}(\xi^{k+1})$$

for all k. (Of course, then whole sequence is determined by ξ^1.)

Let M_1, M_2, ... be a sequence of generalized Turing machines, let φ_1, φ_2 , ... be a hierarchical code for this sequence, let ξ^1 be a hierarchical code configuration for it, where ξ^k is an initial configuration of M_k for each k. Let further be a sequence of mappings Φ^*_1, Φ^*_2, ... be given such that for each k, the pair (φ_{k*}, Φ^*_k), is a simulation of M_{k+1} by M_k. Such an object is called a *tower*. ⌟

The main task of the work will be the definition of a tower, since the simulation property is highly nontrivial.

7 Simulation structure

In what follows we will describe the program of the reliable Turing machine (more precisely, a simulation of each M_{k+1} by M_k as defined above). Most of the time, we will just refer to M_k as M and to M_{k+1} as M^*. Cells will be grouped into colonies, where $Q = B^*/B$ is the colony size. The behavior of the head on each colony simulates the head of M^* on the corresponding cell of M^*. The process takes a number of steps, constituting a *work period.*

Definition 7.1 The parameter $U = T^*/T$ is defined to be twice the maximum number of steps that any simulation work period can take. ⌟

Machine M will perform the simulation even if the noise in which it operates is $(\beta(B,T), \gamma(B^*,T^*))$-sparse. By the above definitions, sparsity means that noise comes in *bursts* that are confined to rectangles of size $\beta(B,T)$ (affecting at most β consecutive tape cells), and are separated from each other in time in such a way that there is at most one burst in any γ neighboring work periods. A design goal for the program the following:

Goal 7.2 (Local correction) *Correct a burst within space and time comparable to its size, and much smaller than the size of a colony work period.*

There are some difficulties faced by our desire for a structured presentation: In order to analyze the error-correcting performance even of one a part of the program, we may need to see the whole, since the noise can bring the machine into some state corresponding to an arbitrary other part.

We mentioned *modes* in Section 4.2. Ordinary simulation proceeds in the normal mode. To see whether the basic structure supporting this process is broken somewhere, each step will check whether the present state is *coordinated* with the currently observed cell symbol (see Definition 8.6). If not then the state jumps into the *healing* mode. We will also say that *alarm* will be called. On the other hand, the state enters into *rebuilding* mode on some indications that healing fails. The crudest outline of the main rule of machine M is given in Rule 7.1; the ***Compute*** and ***Transfer*** rules will be outlined below. (Rebuilding may be triggered inside the ***Heal*** rule.)

Rule 7.1: Main rule

```
if the mode is normal then
    if not Coordinated then Heal
    else if $1 \le Sw <$ TransferSw(1) then Compute
    else if TransferSw(1) $\le Sw <$ Last then Transfer
    else if Last $\le Sw$ then move the head to the new base.
```

7.1 Head movement

The global structure of a work period is this:

Computation phase The new simulated state and direction (called the drift) is computed. Then the "meaningfulness" of the result is checked. During this phase the head sweeps, roughly, back-and-forth between the ends of the base colony.

Transfer phase The head moves into the neighbor colony in the simulated head direction, and transfers the simulated state to there. If the drift is, say, to the right, then the head sweeps, roughly, between the left end of the source colony and the right end of the target colony. There may be an area between the two colonies to bridge over.

The timing is controlled by a field Sw of the state counting the sweeps, and a corresponding field cSw of the cell state. We can read off the direction of the sweep s using the formula

$$\mathrm{dir}(s) = (-1)^{s+1}. \tag{7.1}$$

Some issues complicate the head movement, even in the absence of faults.

7.1.1 Zigging

A burst could turn the head back in the middle of its sweep. To detect such an event promptly (in accordance with Goal 7.2), the sweeping will be complicated by a *zigzag* movement. On its backward zig, the head can check whether the structure of last few cells is correct. On its forward zig, it can also check whether it has not entered into forbidden territory/ Zigging will use certain parameters F, Z that we set to

$$\begin{aligned} F &= 7, \\ Z &= 14F\beta, \\ f_{\mathrm{zig}} = 1, \quad f_{\mathrm{heal}} &= 2, \quad f_{\mathrm{rebuild}} = 3. \end{aligned}$$

The choice of the parameter F will be motivated by the discussion of feathering in Section 7.1.2. In its movement in direction $\delta \in \{-1, 1\}$ the head, every forward sweeping move shifts what is called the *front*. These moves also change the cSw field. Now it will perform a forward-backward zigzag of size approximately $2Z$, without changing anything on the tape. First, it makes $> Z$ more steps forward, looking for the first place ahead where it is allowed to turn back (see Section 7.1.2). As we will see this will be within F steps. Then it will move back to the front and further $> Z$ steps backwards from the front, to the first position where it sill be allowed to turn forward. The fields

$$\mathit{ZigDir} \in \{-1, 0, 1\}, \mathit{ZigDepth} \in [-Z - F, Z + F)$$

control the process. At the front, we have $\mathit{ZigDepth} = 0$ and $\mathit{ZigDir} = 0$. At the start of zigging, we set $\mathit{ZigDir} \leftarrow -1$, and starts *descending* into a (forward) zig, while increasing *ZigDepth*. When the front reaches the extreme edge of zigging at an appropriate turning point (see below), we set $\mathit{ZigDir} \leftarrow 1$, the head turns towards the front and starts *ascending*, while decreasing $|\mathit{ZigDepth}|$. Once the front is reached, a similar backward zigging swing is also performed (with *ZigDepth* decreasing below 0).

7.1.2 Feathering

A somewhat arcane threat is responsible for another complication in head movement.

Example 7.3 Let $C(x)$ denote the colony with starting point x. Consider the following scenario.

1. After the last turn at the end of a right sweep on the right end of colony $C(x)$, at the bottom of the zig, say in position $y = x+(Q+Z)B$ if the zig goes outside the colony, a burst creates some dirt. Then the simulation dictates the head to leave $C(x)$ on the left.
2. Much later, on the first sweep of a return to $C(x)$, this dirt traps the head, and lets it start a right sweep to the right of y. When the head returns on a zig to y, a new burst corrects everything on the left of y, including y, but leaves the start of the new (false) right sweep on the right of y—even though the simulation does not dictate any move to the colony $C(x + QB)$.
3. Much later, the pair of events 1-2 happens again and again, allowing the started false sweep on the right of y to continue. After Z repetitions of this, the zig does not return already.

This way, a number of cleverly placed distant bursts can trigger an effect on the next level. (A probability model more sophisticated than sparsity might deal with this problem, but we did not find it.) ⌟

The sequence of Example 7.3 would not occur if our machines had the following property:

Definition 7.4 (Feathering) A Turing machine execution is said to have the *feathering* property if the following holds. If the head turned back at a position x at some time, the next time the head arrives at position x it cannot turn back from it. ⌟

(The name "feathering" refers to the picture of the path of the head in a space-time diagram.) The property can be enforced by the following simple device: let the cell state have a field called

$$cCanTurn \in \{\mathsf{false}, \mathsf{true}\}.$$

We will allow the head to turn *only* in a cell with $cCanTurn = \mathsf{true}$. Whenever a head turns at a cell, it sets this value to false; if it passes the cell, the value is reset to true. For example, if the head is moving right and decides to turn left then it can do that at the first cell with $cCanTurn = \mathsf{true}$. The following example suggests that any computation can be reorganized to accomodate feathering, without too much extra cost.

Example 7.5 (Feathering) Suppose that, arriving from the left at position 1, the head decides to turn left again. In repeated instances, it can then turn back at the following sequence of positions:

$$1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 6, \dots$$

⌟

So the feathering constraint does not divert the head too much. If in the original execution the head turned back t consecutive times to the left from position 0, then now it will turn back from somewhere in a zone of size $O(\log t)$ to the right of 0 in each of these times. We also see here that the exact turning point at the end of each sweep could be computed from the sweep number.

The simulated Turing machine will also have the feathering property, therefore it will not happen that the simulation turns back repeatedly from the same colony without passing it in the meantime. It may still happen that at a point where it decided to turn, the head encounters a long run of cells with $cCanTurn = \mathsf{false}$. But this indicates the absence of a clean interval comparable with the size of this run—and such cases will have to be dealt with in the context of error correction.

Let us argue, informally (without replacing any later, formal argument), that one does not have to consider more than three islands in any area of size QB. We assume that the simulated program obeys the feathering condition: so if, say, the head comes from the left neighbor colony then at the end of the work period can turn back to the left only if the previous time it visited the colony, it has passed it over from right to left.

Example 7.6 (Three islands) Suppose that the head has arrived at C from the left, performs a work period and then passes to the right. In this case, if no new noise burst occurs then we expect that all islands found in C will be eliminated. On the other hand, a new island I_1 can be deposited (in the last pass). We can assume that there is no island on the left of I_1 within distance QB, since the noise burst causing it would have been too close to the noise burst causing I_1.

Consider the next time (possibly much later), when the head arrives (from the right). If it later continues to the left, then the situation is similar to the above. Island I_1 will be eliminated, but a new one may be deposited. But what if the head turns back left at the end of the work period? If I_1 is close to the left end of C, then due to the feathering construction, the head may never reach it to eliminate it; moreover, it may add a new island I_2 on the right of I_1. We can also assume, similarly to the above, that there is no island on the right of I_2 within distance QB.

When the head returns a third time (possibly much later), from the right, it will have to leave on the left. The islands I_1, I_2 will be eliminated as they are passed over but a possible new island I_3, created by a new burst (before, after or during the elimination), may remain. We can also assume, similarly to the above, that there is no island on the right of I_3 within distance QB. ⌟

Though the *cCanTurn* field assures feathering, we must make sure by organization that it is not preventing any needed turns. For this sake, some further regulations will be introduced. We will make use of a parameter E introduced in Definition 9.3 in connection with the healing procedure. It is a multiple of F.

Zigging in normal mode Above, we made sure that the points for turning from left to right during simulation introduced at the bottom of a zig when sweeping right have $Addr \equiv f_{\text{zig}} \pmod{F}$. This way, during left sweep, the head will always find a place within any interval of F cells with $cCanTurn = \mathsf{true}$; indeed, the zigging turns during the right sweep happened only in one cell in any interval of size F.

Sweeping in normal mode Recall from (4.2) that the colony work period consists of at most V sweeps, and according to Definition 5.6, an area of

$$PadLen = E\lceil \log V \rceil + Z \tag{7.2}$$

cells on both ends of the colony are left free of simulation information—they can be used for feathering. Suppose that the head is to be turned back at "the right end". Then at a distance $PadLen$ from the right end, it starts looking for a place to turn; it will actually turn back at the first position coming after this with $Addr \equiv f_{\text{end}} \pmod{E}$ and $cCanTurn =$ true. This will still allow to find a turning point with $Addr < Q$, as remarked after Example 7.5.

Note that when the head changes direction at the end of a sweep, it has $Addr \equiv f_{\text{end}} \pmod{E}$; in other words, the possible turning points at the colony ends are separated from each other by the larger distance E, a multiple of F. The rationale for this will become clear only when analyzing the boundary of a large clean and a large dirty area.

End turn of healing or rebuilding The healing and the rebuilding procedures need a turning point at the end of their ranges. In case the cell in question has an address (is not a stem cell) then it will need to have $Addr \equiv \pm f_{\text{heal}} \pmod{F}$ for healing turns and $\equiv \pm f_{\text{rebuild}}$ for rebuilding turns.

Zigging while rebuilding Rebuilding mode will have its own internal addresses: as with healing (see later), to the left of the rebuilding front, the addresses are counted from the left end, and to the right of the head from the right end. When sweeping right then the bottom of the left zig will again be at a point with left address $\equiv f_{\text{zig}} \pmod{F}$.

We introduce a convenient notation.

Notation 7.7 Suppose that within a procedure we want the head to move n cells, and turn. It may not be able to turn then, but only at a point with $cCanTurn =$ true and also having for example $cAddr \equiv \pm f_{\text{heal}}$ during healing. For simplicity, we will say that the head moves n^+ cells and turns. Thus n^+ is greater than n, but by not much: only an amount $O(\log \beta)$ for healing, and $O(\log Q)$ for simulation and rebuilding. ⌟

7.2 Computation phase

As shown in Rule 7.1 describing a top-down view of the simulation, the first phase computes new values for the state of the simulated machine M^* represented on track *cState*, the direction of the move of the head of M^* (represented in the *cDrift* field of each cell of the colony of M), and the simulated cell state of M^* represented on the track *cInfo*. During this rule, the head sweeps the base colony.

Recall Definition 5.4. The rule ***Compute*** will rely on a certain fixed $(\beta, 3)$ burst-error-correcting code, moreover it expects that each of the words found on the *cState* and *cInfo* tracks is 2-compliant.

The rule ***ComplianceCheck*** checks whether a word is 2-compliant.

The rule ***Compute*** essentially repeats 3 times the following *stages*: decoding, applying the transition, encoding. Then it calls ***ComplianceCheck***; if the latter fails it will mark the colony for rebuilding.

In more detail:

1. For every $j = 1, \dots, 3$, if $Addr \in \{0, \dots, Q-1\}$ do

   a) Calling by g the string found on the *cState* track of the interior of the base colony, decode it into string $\tilde{g} = \upsilon^*(g)$ (this should be the current state of the simulated machine), and store it on some auxiliary track in the base colony. Do this by simulating the universal machine on the *cProg* track: $\tilde{g} = \mathrm{Univ}(p_{\mathrm{decode}}, g)$.

      Proceed similarly with the string a found on the *cInfo* track of the base colony to get $\tilde{a} = \upsilon^*(a)$ (this should be the observed tape symbol of the simulated machine).

   b) Compute the value $(a', g', d) = \tau^*(\tilde{a}, \tilde{g})$. Since the program of the transition function τ^* is not written explicitly anywhere, this "self-simulation" step needs some elaboration, see Section 7.3.

   c) Write the encoded new state $\upsilon_*(g')$ onto the $cHold[j].State$ track of the interior of the base colony. Similarly, write the encoded new observed cell content $\upsilon_*(a')$ onto the $cHold[j].Info$ track. Write d into the $cHold[j].Drift$ field of *each cell* of the base colony.

      Special action needs to be taken in case the new state g' is a vacant one, that is $g'.Kind^* = Vac^*$. In this case, write 1 onto the $cHold[j].Doomed$ track (else 0).

2. Sweeping through the base colony, at each cell compute the majority of $cHold[j].Info$, $j = 1, \dots, 3$, and write it into the field $cInfo$. Proceed similarly, and simultaneously, with $cState$ and $Drift$.
3. For $j = 1, \dots, 3$, call `ComplianceCheck` on the $cState$ and $cInfo$ tracks, and write the resulting bit into the $Compliant_j$ track.

   Then pass through the colony and turn each cell in which the majority of $Compliant_j$, $j = 1, \dots, 3$ is false, into a stem cell (thus destroying the colony if the result was false everywhere).

It can be arranged—and we assume so—that the total number of sweeps of this phase, and thus the starting sweep number of the next phase, depends only on Q.

7.3 Forced self-simulation

Step 1b of Section 7.2 needs elaboration.

7.3.1 New primitives

We will make use of a special track

$$cWork$$

of the cells and the special field

$$Index$$

of the state of machine M that can store a certain address of a colony.

Recall from Section 4.2 that the program of our machine is a list of nested "**if** *condition* **then** *instruction* **else** *instruction*" statements. As such, it can be represented as a binary string

$$R.$$

If one writes out all details of the construction of the present paper, this string R becomes completely explicit, an absolute constant. But in the reasoning below, we treat it as a parameter.

Let us provide a couple of *extra primitives* to the rules. First, they have access to the parameter k of machine $M = M_k$, to define the transition function

$$\tau_{R,k}(a, q).$$

The other, more important, new primitive is a special instruction

`WriteProgramBit`

in the rules. When called, this instruction makes the assignment $cWork \leftarrow R(Index)$. This is the key to self-simulation: *the program has access to its own bits.* If $Index = i$ then it writes $R(i)$ onto the current position of the $cWork$ track.

7.3.2 Simulating the rules

By convention, in our fixed flexible universal machine Univ, program p and input x produce an output $\mathrm{Univ}(p, x)$. Since the structure of all rules is very simple, they can be read and interpreted by Univ in reasonable time:

Theorem 2 *There is a constant string called Interpr with the property that for all positive integers k, string R that is a sequence of rules, and bit strings $a \in \Sigma_k$, $q \in \Gamma_k$:*

$$\mathrm{Univ}(Interpr, R, 0^k, a, q) = \tau_{R,k}(a, q).$$

The computation on Univ *takes time $O(|R| \cdot (|a| + |q|))$.*

The proof parses and implements the rules in the string R; each of these rules checks and writes a constant number of fields.

Implementing the ***WriteProgramBit*** instruction is straightforward: Machine Univ determines the number i represented by the simulated $Index$ field, looks up $R(i)$ in R, and writes it into the simulated $cWork$ field.

Note that there is no circularity in these definitions:

- The instruction ***WriteProgramBit*** is written *literally* in R in the appropriate place, as "***WriteProgramBit***". The string R is *not part* of the rules (that is of itself).
- On the other hand, the computation in $\mathrm{Univ}(Interpr, R, 0^k, a, q)$ has *explicit* access to the string R as one of the inputs.

Let us show the computation step invoking the "self-simulation" in detail. In the earlier outline, step 1b of Section 7.2, said to compute $\tau^*(\tilde{a}, \tilde{g})$ (for the present discussion, we will just consider computing $\tau^*(a, q) = \tau_{k+1}(a, q)$), where $\tau = \tau_k$, and it is assumed that a and q are available on two appropriate auxiliary tracks. We give more detail now of how to implement this step:

1. Onto the *cWork* track, write the string R. To do this, for *Index* running from 1 to $|R|$, execute the instruction ***WriteProgramBit*** and move right. Now, on the *cWork* track, replace it with $\langle \mathit{Interpr}, 0^{k+1}, R, a, q\rangle$. Here, string *Interpr* is a constant, so it is just hardwired. String R already has been made available. String 0^{k+1} can be written since the parameter k is available. Strings a, q are available on the tracks where they were stored.
2. Simulate the universal automaton Univ on track *cWork*: it computes $\tau_{R,k+1}(a, q) = \mathrm{Univ}(\mathit{Interpr}, R, 0^{k+1}, a, q)$ as needed.

This achieves the forced self-simulation. Note what we achieved:

- On level 1, the transition function $\tau_{R,1}(a, q)$ is defined completely when the rule string R is given. It has the forced simulation property by definition, and string R is *"hard-wired"* into it in the following way. If $(a', q', d) = \tau_{R,1}(a, q)$, then

$$a'.\mathit{cWork} = R(q.\mathit{Index})$$

whenever $q.\mathit{Index}$ represents a number between 1 and $|R|$, and the values $q.\mathit{Sw}$, $q.\mathit{Addr}$ satisfy the conditions under which the instruction ***WriteProgramBit*** is called in the rules (written in R).

- The forced simulation property of the *simulated* transition function $\tau_{R,k+1}(\cdot, \cdot)$ is achieved by the above defined computation step—which *relies on* the forced simulation property of $\tau_{R,k}(\cdot, \cdot)$.

Remark 7.8 This construction resembles the proof of Kleene's fixed-point theorem. ⌟

7.4 Transfer phase

In the transfer phase, simulated state information will be transferred to the neighbor colony in the direction of the simulated head movement: this is called the direction of the transfer, or the *drift*. During this phase, the range of the head includes the base colony and the neighbor colony determined by the drift, including a possible bridge between them.

The sweep number in which we start transferring in direction δ is called $\mathrm{TransferSw}(\delta)$, the *transfer sweep*. We have $\mathrm{TransferSw}(-1) = \mathrm{TransferSw}(1) + 1$.

7.4.1 General structure of the phase

We will make use of some extra rules that we will specify in more detail later, but whose role is spelled out here.

The phase consists of the following actions.

1. Spread the value δ found in the cells of the *cDrift* track (they should all be the same) onto the neighbor colony in direction δ.

   There are some details to handle in case the neighbor colony is not adjacent: see Section 7.4.2.

2. For $i = 1, 2, 3$:

   > Copy the content of *cState* track of the base colony to the *cHold*$[i]$.*State* track of the neighbor colony.

3. Repeat the following twice:

   > Assign the field majority: $\mathit{cState} \leftarrow \operatorname{maj}(\mathit{cHold}[1\dots3].\mathit{State})$ in all cells of the neighbor colony.

4. If $\mathit{Drift} = 1$, then move right to the left end cell of the neighbor colony (else you are already there).

5. In the last sweep (possibly identical with the move step above), in the base colony, if the majority of *cHold*$[j]$.*Doomed*, $j = 1, \dots, 3$, is 1 then turn the scanned cell into a stem cell: in other words, carry out the destruction.

7.4.2 Transfer to a non-adjacent colony

Let us address the situation when the neighbor colony is not adjacent.

Definition 7.9 (Adjacency of cells) Cells a and b are *adjacent* if $|a-b| = B$. Otherwise, if $B < |a-b| < 2B$, then a and b are two *non-adjacent neighbor cells*. For the sake of the present discussion, a *colony* is a sequence of Q adjacent cells whose *cAddr* value runs from 0 to $Q-1$. It may be extended by a bridge of up to $Q-1$ adjacent cells in the direction of the drift.

If the bodies of two cells are not adjacent, but are at a distance $< B$ then the space between them is called a *small gap*. We also call a small gap such a space between the bodies of two colonies. On the other hand, if the distance of the bodies of two colonies is $> B$ but $< QB$ then the space between them is called a *large gap*. ⌟

In the transfer phase, in order to know in a robust local way where the head is, the *cKind* field of the cells visited will be set as follows. The base colony has cells of kind Member to begin with. The kind of the cells of the neighbor colony, the target of the transfer, will be set as Target for the duration of the transfer. However, in the first transfer sweep, if there was a gap between the base and the target, then cells between them will be created or adapted to form a bridge that extends the base colony, also extending its addresses. It is added to the workspace. A bridge can override an opposite old bridge ("old" meaning that its Sw is maximal) or move into an empty area, or kill opposite bridge cells or stem cells while it extends. If while forming a bridge, another colony is encountered before the bridge grows to length QB), then this new colony's cells will get the kind Target, and all will be added to the workspace. (There can be a gap of size $< B$ between the bridge and the neighbor colony.) Otherwise the bridge itself becomes this neighbor colony, and its cell kinds are turned to Target on the return sweep.

Recall that the *NonAdj* field of the state determines if the current cell is not adjacent to the cell where the head came from. After the transfer stage, we update the $NonAdj^*$ field encoded in the *cState* track of the target colony: it becomes 1 if either there is a nonempty bridge, or there is a gap (found with the help of the *NonAdj* field) between the base colony and the target colony. This is done in part 2 of Section 7.4.1 again three times, storing candidate values into $cHold[j].NonAdj$ and repeated with everything else.

8 Health

The main part of the simulation uses an error-correcting code to protect information stored in *cInfo* and *cState* fields. However, faults can ruin the simulation structure and disrupt the simulation itself. The error-correcting capabilities of the code used to store the information on the *cInfo* and *cState* tracks, will preserve the content of these tracks as long as the coding-decoding process implemented in the simulation is carried out. The structural integrity of a configuration is maintained with the help of a small number of fields. Below we outline the necessary relations among them allowing the identification and correction of local damage.

A configuration with local structural integrity will be called *healthy*.

Remark 8.1 In all discussions of the health of a configuration $\xi = (q, A, h, \hat{h},)$, we can ignore the current cell position $\hat{h} = \xi.\text{cur-cell}$, since at all switching times it agrees with $h = \xi.\text{pos}$. ⌟

No cell in a healthy configuration should have marks of a rebuild procedure. Larger bursts may also introduce new, non-local anomalies: these can only be recognized once the local anomalies have been corrected. Cells of a healthy configuration are grouped into gapless colonies, and a few transitional segments described below. The big picture is this:

- There is a base colony, possibly extended by a bridge in the direction of the drift. Possibly, in the direction of the drift, the neighbor colony of the base is a *target*, its cells are marked as such.
- Non-base colonies are called *outer colonies*. If an outer colony is not adjacent to its neighbor colony closer to the base, then it is extended by a bridge that covers this gap.

  A partial exception is the colony C closest to the base colony in the direction of the drift. If there is a gap between it and the base colony then initially it is covered by a bridge extending C, but in the first transfer sweep this bridge will be overridden by the bridge extending the base colony.

⟪P: Pictures!⟫

Definition 8.2 (Outer cells) Recall the definition of the sweep value $\text{Last}(\delta)$ from (4.1). For $\delta \in \{-1, 1\}$, if a cell is stem or $cDrift = \delta$, $cSw = \text{Last}(\delta)$ then it will be called a *right outer cell* if $\delta = -1$, $-Q < cAddr < Q$, and a *left outer cell* if $\delta = 1$, $0 \le cAddr < 2Q - 1$. ⌟

According to this definition, a stem cell is both a left and a right outer cell.

Definition 8.3 (Segments) The following possible sequences of neighbor cells will be called a *(homogenous) segment*:

Desert A sequence of neighboring (not necessarily adjacent) stem cells.

Workspace segment A sequence of adjacent neighbor cells that could form part of the workspace and does not include the front. It consists either of only target cells or of cells of an extended base colony.

Outer segment A sequence of outer cells that could belong to the same extended colony.

The *left end* of a segment is the left edge of its first cell, and its *right end* is the right edge of its last cell.

A *colony* has addresses grow from 0 to $Q-1$, possibly continued by a bridge of size $< Q$ on one side, and a right end segment. It is a *target* if it consists of target cells. A target is never extended by a bridge.

A *boundary* of a homogenous segment is called *rigid* if its address is the end address of a colony in the same direction.

A *boundary pair* is a right boundary followed by a left boundary at distance $< B$. It is a *hole* if the distance is positive. It is *rigid* if at least one of its elements is. ⌟

In a healthy configuration, cells fall into certain categories. Outer cells are member cells in colonies other than the ones that are currently being manipulated.

Recall the definition of the *transfer sweep* $\mathrm{TransferSw}(\delta)$ in Section 7.4, if $\delta \neq 0$. (There is no transfer sweep if $\delta = 0$.)

Definition 8.4 (Healthy configuration) The health of a configuration ξ of a generalized Turing machine M will be defined over a certain interval A. It depends on the state, on $\xi_{\mathrm{tape}}(A)$, further on whether ξ_{pos} is in A and if it is, where. But we will mention the interval A explicitly only where it is necessary. In particular, some of the structures described below may fall partly or fully outside A. We require that mode be normal, and the following conditions hold, with $\delta = \mathit{Drift}$.

Normality No cell in A is marked for rebuilding (recall the notion of marking from Section 4.2): that is, for every $x \in A$, $\mathit{Rebuild}.\mathit{cSw}(x) = 0$.

Segments The *base* is defined by counting back from Addr. This is simple as long as we are within one colony. However, when we are passing from a target cell to a member cell, then addresses on the tape will undergo a *jump*: we set Addr to cAddr before continuing the count-back.

All cells can be grouped into full extended colonies, with possibly some stem cells between these. In more detail:

- An *extended base colony* consisting of member cells and bridge cells.
- *Extended outer colonies*, consisting of outer member cells.
- A possible target, defined by the value of cSw in its cells.
- Desert filling out the gaps between the above parts.

To define the non-base segments, we consider several cases.

- If $\delta = 0$ or $Sw < \mathrm{TransferSw}(\delta)$, then there are no bridge or target cells.
- If $\mathrm{TransferSw}(\delta) + 1 < Sw <$ (the last two sweeps for δ), then there is a target colony in the direction δ. If its distance from the base colony is $\geq B$ then the base colony is extended by a bridge filling the gap.
- If $Sw = \mathrm{TransferSw}(\delta)$ then the above described situation is in the making, as a bridge is being built up in direction δ, or after that, a target is being built in direction δ, converting member cells into target cells.
- If $Sw = \mathrm{TransferSw}(\delta) + 1$ and there is still not a complete target colony, then the whole bridge is being converted into a target, as the head is traveling in direction $-\delta$.
- In the last sweep, the target cells are being converted into member cells.

These are the only possible segments to be seen in a healthy area.

The front The farthest position $\mathrm{front}(\xi)$ to which the head has advanced before starting a new zig is called the *front*: it can be computed from the fields $\xi.\mathrm{pos}$ and *ZigDepth* of the state, but can also be reconstructed from the tape, namely from the *cSw* track. It is always inside the extended base colony or the target.

Workspace The *workspace* is an interval of non-outer cells, such that:

- For $Sw < \mathrm{TransferSw}(\delta)$, it is equal to the base colony.
- In case of $Sw = \mathrm{TransferSw}(\delta)$, it is the smallest interval including the base colony and the cell neighboring to $\mathrm{front}(\xi)$ on the side of the base colony.
- If $\mathrm{TransferSw}(\delta) < Sw < \mathrm{Last}(\delta)$, then it is equal to the union of the extended base colony and the target.
- When $Sw = \mathrm{Last}(\delta)$, it is the smallest interval including the target (future base) colony and $\mathrm{front}(\xi)$.

A tape configuration is called *healthy* on an interval A when there is a head position (possibly outside A) and a state that turns it into a healthy configuration. ⌟

Definition 8.5 Let the tuple of fields

$$Core = (Addr, Sw, Drift, Kind)$$

is called the *core*. ⌟

Note that health only depends on the fields in *Core* and the zigging field Z in the state, further the *cCore* track and the lack of marks on the tape. Note also that in a healthy configuration every cell's *cCore* field determines the direction in which the front is found, from the point of view of the cell.

A violation of the health requirements can sometimes be noted immediately:

Definition 8.6 (Coordination) The state of the machine is *coordinated* with the current cell if it is possible for them to be together in a healthy configuration. ⌟

Recall that in Rule 7.1, in normal mode, if lack of coordination is discovered then the healing procedure is called. The following lemmas show how local consistency checking will help achieve longer-range consistency.

Lemma 8.7 *In a healthy configuration, each Core value along with Z determines uniquely the cCore value of the cell it is coordinated with, with the following exceptions.*

- *During the first transferring sweep, while creating a bridge between the base colony and the target colony, the front can be a stem cell or the first cell of an outer colony.*
- *Every jump backward from the target colony can end up on the last cell of a bridge (whose address is not recorded in the state) or the last cell of the base colony.*

Proof. To compute the values in question, calculate Z steps backwards from the front, referring to the properties listed above. ⟨⟨P: Elaborate!⟩⟩ □

Lemma 8.8 (Health extension) *Let ξ be a* tape *configuration that is healthy on intervals A_1, A_2 where $A_1 \cap A_2$ contains a whole cell body of ξ. Then ξ is also healthy on $A_1 \cup A_2$.*

Proof. The statement follows easily from the definitions. ⟨⟨P: Elaborate!⟩⟩ □

In a healthy configuration, the possibilities of finding non-adjacent neighbor cells are limited.

Lemma 8.9 *An interval of size $< Q$ over which the configuration ξ is healthy contains at most two maximal sequences of adjacent non-stem neighbor cells.*

Proof. Indeed, by definition a healthy configuration consists of full extended colonies, with possibly stem cells between them. An interval of size $<$

Q contains sequences of adjacent cells from at most two such extended colonies. □

Lemma 8.10 *In a healthy configuration, a cell's content shows whether it is an outer cell, colony cell, bridge cell or target cell. It also shows whether the cell is to the left or to the right of the front.*

The cCore track of a homogenous segment can be satisfactorily reconstructed from its endcells. The reconstruction unique, with one exception: in case the segment is not internal: the cSw values are not uniquely defined, only the parity and the direction of its change between the ends.

Proof. The information mentioned in the first sentence is explicit in some fields. About the front: if the cell is outer then *cDrift* shows its direction from the front. In case it is not outer, then the parity of *cSw* shows it: odd values are on the left, even ones on the right.

The uniqueness of reconstruction is a direct consequence of the definitions. □

9 Healing and rebuilding

9.1 Annotation, admissibility, stitching

In this section we show how to correct configurations of machine M that are "almost" healthy.

Definition 9.1 (Annotation) An *annotated configuration* is a tuple

$$(\xi, \chi, \mathcal{I}),$$

with the following meaning.

ξ is a configuration.

$\mathcal{I}$ is a set of disjoint intervals called *islands*. Their complement is clean.

χ is a healthy configuration differing from ξ only in the islands.

The *base colony* and the *workspace* of $(\xi, \chi, \mathcal{I})$ are those of χ. The head is *free* when it is not in any island, and the state is in normal mode, coordinated with the observed cell. ⌟

Definition 9.2 (Admissibility) An annotation is *admissible* on an interval K if the following holds on any subinterval of $J \subseteq K$ of size QB:

a) The dirt of J is covered by 3 intervals, each of size $\le \beta' B$.

b) There are at most 3 islands in J, each of size at most $c_{\text{island}}\beta B$.

A configuration is *admissible* on K if it can be annotated in a way admissible on K. ⌟

We will show that a configuration admissible over an interval of a certain size can be locally corrected; moreover, in case the configuration is clean then this correction can be carried out by the machine M itself. For the time being, we will just talk about an admissible configuration, without specifying the interval K.

We will deal with the cleaning process later, but it will imply that, in the absence of new noise, islands will not grow much, and the ones near the head will be eliminated.

Definition 9.3 (Substantial segments) Let $\xi(A)$ be a tape configuration over an interval A. A homogenous segment of size at least $7c_{\text{island}}\beta B$ will be called *substantial*. The area between two neighboring maximal substantial segments or between an end of A and the closest substantial segment in A will be called *ambiguous*. It is *terminal* if it contains an end of A. Let

$$\begin{aligned}\Delta &= 39c_{\text{island}}\beta,\\ E &\geq 6\Delta.\end{aligned}$$

We assume that E is an integer multiple of the parameter F introduced for feathering in Section 7.1.1. ⌟

Lemma 9.4 *Consider an admissible configuration. In a substantial segment, each half contains at least one cell outside the islands.*

If an interval of size $\leq QB$ of a tape configuration ξ differs from a healthy tape configuration χ in at most three islands, then the size of each ambiguous area is at most ΔB.

Proof. The first statement is immediate from the definition of substantial segments There are at most 3 boundary pairs in χ at a total size of $3B$, and 3 islands of size $\leq c_{\text{island}}\beta B$. There are at most 5 non-substantial segments of sizes $< 7c_{\text{island}}\beta B$ between these: this adds up to

$$< (3 + 3 \cdot c_{\text{island}}\beta + 5 \cdot 7 \cdot c_{\text{island}}\beta)B < 39 \cdot c_{\text{island}}\beta B = \Delta B.$$

□

The following lemma forms the basis of the cleaning algorithm.

Lemma 9.5 (Stitching) *In an admissible configuration, inside a clean interval, let U, W be two substantial segments separated by an ambiguous area V. It is possible to change the cell content U, V, W using only information in U, W in such a way that the configuration over $U \cup V \cup W$ becomes healthy. Moreover, it is possible for a cellular automaton to do so gradually, keeping admissibility, and changing the cell content in U or W or enlarging U or W gradually at the expense of V.*

We do not worry about the feathering property now, since the cellular automaton in question may choose where to make its turns.

Proof. We distinguish several cases, based on the kind of segments involved. At any step, if we find that $U \cup V \cup W$ is healthy then we stop.

1. Assume that U, W both belong to the same extended colony: either the extended base colony, or an outer extended colony, or the target.

   If both belong to the interior of the area indicated, then the merging is simple. In this case, by Lemma 8.10, the content of the *cCore* track of V in the healthy configuration is completely determined by that of U, W, So the intervals U and W can be gradually extended towards each other in any order, overtaking V.

   Suppose that they do not belong to the interior and are, for example towards the right from it. In this case, the *cSw* value may be decreasing in both U and W to the right. But since U, W themselves may overlap with islands, it may happen that the *cSw* value on the right end of U is smaller than the *cSw* value on the left end of W. In this case, before extending U, W towards each other, the *cSw* values in both may need to be changed.

   By Lemma 9.4, the left and right halfs of U contains a cell u_1, u_3 each, not belonging to any island, and similarly with w_1, w_3 in W. Let u_2 be leftmost cell of the right half of U and w_2 the rightmost cell of the left half of W. Then

   $$cSw(u_1) \geq cSw(u_2) \geq cSw(u_3) \geq cSw(w_1) \geq cSw(w_2) \geq cSw(w_3).$$

   So the transformation will change *cSw* to $cSw(u_2)$ everywhere in the right half of U, and to $cSw(w_2)$ everywhere in the left half of W. After this, it will extend U towards W, overtaking V and keeping *cSw* constant.

   In both of the above cases, if U and W belonged to different sides of the front, then the new front will be the new boundary between U and W.

2. Suppose now that U, W do not belong to the same extended colony, but they are on the same side of the front: without loss of generality, to the left of it.

   In this case, only one of U can be outer: suppose that it is. Now W is either in a target or in the base colony. The base colony cannot have an extension to the left, because given that the front is to the right, the same sweep that created the extension would have created already a target, separating U from W too much. So in both cases, W is close to the left end of its colony which is in V. It must be extended to this left end. Following this, U must be extended until its reaches W. The cSw values can extended without change, there is no need to coordinate them between U and W.

   Suppose that U is a target, and then W belongs to the extended base colony. Since both U and W are in the interior, the cSw values must be equal, so they can be extended without change. Then U must be extended until its endcell, and then W must be extended to meet U. Since both U and W are in the interior, the cSw values must be equal, so they can be extended without change.
3. Suppose that U and W belong to different extended colonies, with the front between them.

   The cSw values can extended without change, there is no need to coordinate them between U and W.

   Suppose that U contains no bridge cells: in this case extend it to the right until it hits a colony endcell. If you overlap W, decrease W accordingly. Due to admissibility, only bridge cells of W can be overwritten in this way. Similarly, if W contains no bridge cells then extend it to the left, until it hits a colony endcell. Again, only bridge cells of U can be overwritten in this way.

   Only one of U and W can be in an outer extended colony, suppose that U is. If W is a target then we already extended it to its limit. Now extend the bridge of U to meet it. If W is in an extended base colony then extend its bridge until either meets U or reaches full length. In the latter case, extend the bridge of U to meet W.

   If U is in a target then it is already at its right end: we extend the bridge of W to meet it.

$\square$

9.2 The healing procedure

Structure repair will be split into two procedures. The first one, called *healing*, performs only local repairs of the structure: for a given (locally) admissible configuration, it will attempt to compute a satisfying (locally) healthy configuration. If it fails—having encountered a configuration that is not admissible, or a new burst—then the *rebuilding* procedure is called, which is designed to repair a larger interval. On a higher level of simulation, this corresponds to the implementation of the "cleaning" trajectory properties. The healing procedure runs in $O(\beta)$ steps, whereas rebuilding needs $O(Q^2)$ steps. ⟪P: maybe even Q^3?⟫

The description of the procedures looks as if we assumed that there is no noise or dirt. The rules described here, however (as will be proved later), will clean an area locally under the appropriate conditions, and will also work under appropriately moderate noise.

The healing procedure does not even protect itself against any possible noise during it. The only protection is that any one call of the healing procedure will change only a small part of the tape, essentially one cell: so a noise burst will have limited impact even if it happens during healing.

One possible outcome of the healing procedure is *failure*. In this case the plan is to mark a "germ" of 4β cells and call rebuilding. The healing procedure carries out only one step of this plan, then it calls itself again.

Recall the parameters Δ, E introduced in Definition 9.3. Suppose that ***Heal*** is called at some position z. Then it sets

$$\mathit{Mode} \leftarrow \text{Healing},\ \ \mathit{Heal.Sw} \leftarrow 1,\ \ \mathit{Heal.Addr} = 0.$$

The healing procedure starts by surveying an interval R of at least $2E$ cells around its starting point z. (It will go a little farther than E cells, in search of an allowed turning point. If such a point is not found within $3\beta + F$ steps, healing fails.) Whenever the head steps on a stem cell or creates a new cell, *cDrift* is set to point to z (to make sure that the head does not get lost in the desert).

Suppose that in the survey a pair of substantial segments separated by an ambiguous area is found, that is at least Δ removed from the complement of R. Choose the ambigous area closest to the center (of healing). Then the first needed "stitching" operation (a single step) as defined in Lemma 9.5 is determined, and is performed. If the ambigous area still remains, go to its leftmost cell. and restart healing. In case that we started from a clean admissible configuration, this operation creates a new admissible configuration that is one step closer to being healthy.

If the required stitching operation is not found then, if the surveyed area is found inadmissible, healing fails. Otherwise, if the head is at the front, healing is declared completed. Otherwise, the healing center is moved one step closer to the front, and healing is restarted.

The healing operation defined this way has the following property.

Lemma 9.6 *Assume that the head moves in a noise-free and clean space-time rectangle $[a, b) \times [u, v)$, with $b - a > 3EB$, $v - u > 2ET$, touching every cell of $[a, b)$ at least once, and never in rebuilding mode. Then at time v, the area $[a + 1.5EB, b - 1.5EB)$ is healthy.*

Proof. In healing, the head will not move away from an ambiguous area before eliminating it, creating a healthy interval I containing the two substantial intervals that originally bordered it. If the head leaves this interval in normal mode and the zigging does not start new healing then the healthy interval covered by zigging can be added to I to form an even larger healthy interval, and this continues until new healing starts. One of the substantial intervals of the new healing will be inside I, and when it is completed,I will be extended further. The head may later return into the healthy interval I, but it will then just continue the simulation without affecting health while stying inside. □

9.3 Rebuilding

If healing fails, it calls the rebuilding procedure. This indicates that the colony structure is ruined in an interval of size larger than what can be handled by local healing. Just as with healing, we will be speaking here only about a situation with no mention of dirt—but the final analysis will take dirt into account.

Since rebuilding makes changes on an interval of the length of several colonies, it is important not only that it is invoked when it is needed, but that it is not invoked otherwise! A failed healing ends on a “germ” of cells marked for rebuilding, with the state in rebuild mode. Rebuilding starts by extending the germ to the left, in a zigging way, expecting rebuild-marked cells on the backward zig. Since the zig is larger than a burst, if rebuilding started from just a burst then this will trigger healing, thus leaving the rebuilding mode.

Here is an outline, for rebuilding that started from a cell z in the middle of the germ. Recall the stitching operation from Section 9.1.

Mark Starting from the germ, extend a rebuilding area over $3Q$ cells to the left and $3Q$ cells to the right from z. Mark the area using the addresses $\mathit{Rebuild.cAddr}_j$ for $j = \pm 1$, where addresses are counted from both ends of the rebuilding area, and the track $\mathit{Rebuild.cSw}$. Use zigging to make sure that false rebuilding started by a burst will be recognized (since we started from a germ, the zigging must always see more than β marked cells already present). Also in what follows, every step that changes the configuration must be accompanied by zigging, to check that the rebuilding is indeed going on.

Survey and Create The details of this stage will be outlined below. It looks for existing colonies (possibly needing minor repair) in the rebuilding area, and possibly creates some. As a result, we will have one colony called C_{left} on the left of z, one called C_{right} on the right of z, and possibly some colonies between them. Make all these colonies represent stem cells. Declare C_{left} the base colony, direct all the others with drifts and bridges towards it. (The creation of a bridge may result also in the creation of a new colony if the bridge becomes Q cells long.) Survey and possibly change additional $0.5Q$ cells on the left of C_{left} and $0.5Q$ cells on the right of C_{right}, making the whole interval healthy. This interval will be called the *output interval* of rebuilding.

Mop Remove the rebuild marks, shrinking the rebuilding area onto the left end of C_{left}.

Details of the Survey and Create stage

s1. In the marked area for substantial segments, search on left of z for a colony C, or a set of cells that looks stitchable by up to 3 stitches into a colony. The first substantial segment should be at least 3β cells to the left of z, to make sure that C is *manifestly* to the left of z. If the search and the stitching attempt are successful, mark C as C_{left}.

Repeat this search for a colony manifestly to the right of z: if found, call it C_{right}.

s2. Suppose that only C_{left} is found, then create C_{right} The other case is symmetrical.

s3. Suppose that neither C_{left} nor C_{right} have been found. Then search the whole area, starting from the left, for a colony. If no such colony is found then create C_{left} and C_{right}.

s4. Suppose that both C_{left} and C_{right} have been determined (found or created). Then search between them, from the left, for (stitcheable) colonies, one-by-one.

s5. Suppose that only a colony has been found that is not manifestly to the left or right of z; then either the left end is manifestly to the left or the right end is manifestly to the right of z.

Suppose that the left end it is manifestly to the left of z, then call the colony C_1. Then create C_{left} on the left of C_1. Now search for another one on its right (it is not manifestly on the right). If not found, create C_{right}, manifestly on the right of z. If found call it C_2, and create C_{right} on its right. The other case is symmetrical.

The steps above requiring the creation of colonies and bridges are destructive (the stitching ones are not). Therefore before a creation step, the whole survey preceding it is performed twice, marking the result in all rebuilding cells. The required actions (erasing cells in order to build new ones in their place) are then only performed if the two survey results are identical—otherwise alarm is called.

Rebuilding also obeys feathering: marked turning points allowed for rebuilding must have $\mathit{Rebuild.cAddr} \equiv \pm f_{\text{rebuild}} \pmod{F}$.

How to defend against the effects of a burst during the rebuild procedure? There is only limited defense. The major decisions on creation are made twice, with the two results compared. Otherwise if the usual zigging (with simple consistency check on the rebuild addresses and sweeps) fails, healing is called. This will most likely start another rebuilding, which now will operate without a burst. Traces (say, marked cells) from the first, interrupted rebuilding might remain, and trigger a new heal-rebuild cycle if found at the mopping stage.

The following lemma is an immediate consequence of the definition of rebuilding.

Lemma 9.7 *Suppose that a rebuilding procedure runs noiselessly from start to finish in a clean interval I, starting on the boundary of a subinterval $J \subset I$ that is super-healthy at the beginning. If K denotes the output interval of rebuilding, then $J \cup K$ will be healthy. The part spanned from the leftmost to the rightmost colony will be super-healthy.*

10 Super annotation and scale-up

It is convenient to introduce some additional structures when discussing the effects of moderate noise and their repair.

10.1 Super annotation

As indicated in Section 6, when dealing with the behavior of machine M over some space-time rectangle, we will assume that the noise over this rectangle is $(\beta(B,T),\gamma(B^*,T^*))$-sparse. With Definition 7.1 of T^* this means in simpler terms that at most one noise *burst* affecting an area of size at most βB can occur in any γ consecutive work periods. In the present section, histories will always be assumed to have this property.

Some histories lend themselves to be viewed as a healthy development that is disturbed only in some well-understood ways. Added to such histories the information pointing out these disturbances will be called an annotation. The proof of the error-correcting behavior of machine M (essentially the proof of the Transition Function property of trajectories of Definition 6.7 for the simulated machine M^*) will take the form of showing the possibility of annotation under sparse of noise. An annotation, as per Defnition 9.1, marks some ways in which the health of a tape configuration has been be affected. Now extend annotation in order to deal with damage not only to health but also to information.

Definition 10.1 (Super healthy) A configuration is *super healthy* if in addition to the requirements of health, in each colony, whenever the head is not in the last sweep, the *cInfo* and *cState* tracks contain valid codewords as defined in Section 5.2. A configuration ξ defined on an interval I is *(super) healthy* on I if it can be extended to a (super) healthy configuration. ⌟

Definition 10.2 (Super annotation) A *super annotated configuration* is a tuple

$$(\xi,\chi,\mathcal{I},\mathcal{S}),$$

with the following meaning.

$(\xi,\chi,\mathcal{I})$ is an annotated configuration.

$\mathcal{S}$ is a set of disjoint intervals called *stains*. All islands are contained in stains.

We can change χ into a super healthy configuration by changing it only in the stains. ⌟

Recall Definition 9.2 of admissibility.

Definition 10.3 (Super admissibility) A super annotation is *super admissible* on an interval K if it is admissible on K, further consider any interval $J \subseteq K$ of size $\le QB$.

c) At most 3 stains intersect J.

d) If J contains k stains, surrounded by some healthy area (we already know $k \le 3$), while the head is at a distance $> 2B$ within the clean area, then the total size of these stains is at most $k \cdot c_{\text{stain}} B$, where

$$c_{\text{stain}} = (F+2)\beta. \tag{10.1}$$

e) If a colony in J outside the workspace intersects two stains then the simulated cell state decoded from it has $cCanTurn = \mathsf{false}$.

⌟

Let us motivate requirement 10.3.e. One stain can arise and remain somewhere for example if a burst occurs in the last sweep of a work period at the bottom of a zig. A second stain can remain at the end of a work period in which the simulated machine makes a turn—this sets $cCanTurn \leftarrow \mathsf{false}$ in the simulated cell. Requirement 10.3.e says that in an annotated configuration, this is the only way for two stains to remain outside the workspace.

A configuration may allow several possible super annotations; however, the valid codewords (referred to in the definition of super health) that can be recovered from it do not depend on the choice of the annotation.

The following definitions help extend the notion of annotation to histories.

Definition 10.4 (Distress and relief, safety) Consider a sequence of annotated configurations over a certain time interval. If the head is free (see Definition 9.1), then the time (and the configuration) will be called *distress-free*. A time that is not distress-free and is preceded by a distress-free time will be called a *distress event*. This can be of two kinds: the head steps onto an island, or a burst occurs (creating an island and leaving the head in it).

Consider a time interval K starting with a distress event and ending with a distress-free configuration. Let J be the interval of tape where the head passed during K, then we will call $J \times K$ a *relief event*, if the following holds.

a) Any new islands occurring in $J \times K$ are due to some new burst.

b) The island that started the distress event disappears by the end of K.

⌟

In a relief event, it is possible to leave behind some islands other than the one initiating the distress if the sweeping direction changes during it. Consider the situation in Example 7.6. In the work period where the head deposits island I_2 near the left colony end, it may repeatedly dip into I_2 at the descending end of a zig at a right turn. During this dip it can expand the islands I_1, I_2 and then emerge on the right, with the healing unfinished.

We will consider the annotation of histories over a limited space-time region, but will not point this out repeatedly.

Definition 10.5 (Annotated history) An *annotated history* of a generalized Turing machine

$$M = (\Gamma, \Sigma, \tau, q_{\text{start}}, F, B, T)$$

is a sequence of *super* annotated configurations such that the sequence of underlying configurations is a trajectory, and every distress event is followed by a relief event $J \times K$ with $|J| = O(\beta B)$ and $|K| = O(\beta^2 T)$. ⌟

In what follows we will show that for any trajectory (η, Noise) of a generalized Turing machine M on any space-time rectangle on which the noise is $(\beta(B,T), \gamma(B^*,T^*))$-sparse, if at the beginning the configuration was super healthy then the history can be annotated. In the rest of the section we always rely on the assumption of this sparsity property of the noise.

The main part of the proof is about obtaining relief after a distress event. Unlike in [1], now islands may have their cell structure damaged: may contain dirt. However, since η is a trajectory, as we will see the islands will be cleaned out. So, relief will be made up of two stages: cleaning, and correcting the structure. This division is only possible for an observer: the machine has no “dirt-detector”, we just rely on the cleanness-extending properties of a trajectory introduced in Definition 6.7.

10.2 The simulation codes

Let us now define formally the codes φ_{*k}, Φ^*_k that are needed for the simulation of history $(\eta^{k+1}, \mathit{Noise}^{(k+1)})$ by history $(\eta^k, \mathit{Noise}^{(k)})$. Omitting the index k we will write φ_*, Φ^*. To compute the configuration encoding φ_* we proceed first as done in Section 6.3, using the code ψ_* there, and then add some initializations: In cells of the base colony and its left neighbor colony, the sweep and drift fields *cSw* and *cDrift* are set to $\mathrm{Last}(1) - 1$, 1, and $\mathrm{Last}(1)$, 1 respectively. In the right neighbor colony, these values are $\mathrm{Last}(-1)$ and -1 respectively. In all other cells, these values are empty. The *cAddr* fields of each colony are filled properly: the *cAddr* of the j cell of a colony is $j \bmod B^*$. ⟪P: Picture?⟫

The value $\mathit{Noise}^{(k+1)}$ is obtained by a residue operation just as in Definition 5.2 of sparsity. It remains to define $\eta^* = \eta^{(k+1)}$ when $\eta = \eta^k$. Parts of the history that are locally super-annotated will be called clean. In the clean part, if no colony has its starting point at x at time t, set $\eta^*(x,t) = \mathit{Vac}$. Otherwise $\eta^*(x,t)$ will be decoded from the *cInfo* track of this colony, at the beginning of its work period containing time t. More precisely:

Definition 10.6 (Scale-up) Let (η, Noise) be a history of M, where $\mathit{Noise} = \mathit{Noise}^{(k)}$ as in Definition 5.2. We define $(\eta^*, \mathit{Noise}^*) = \Phi^*(\eta, \mathit{Noise})$ as follows. Let $\mathit{Noise}^* = \mathit{Noise}^{(k+1)}$. Consider position x, and let $I = [x - 2.5QB, x + 2.5QB)$, $J = (t - T^*, t]$. If the history (η, Noise) cannot be super-annotated on $I \times J$ then $\eta^*(x,t) = \mathit{Bad}^*$; assume now that it is, and let $\chi(\cdot, u)$ be some super healthy configuration satisfying η over I at time u. If in I there is not a full colony of χ starting on left of x and also a full colony ending on the right of x then $\eta^*(x,t) = \mathit{Bad}^*$. Else if x is not the start of a colony then let $\eta^*(x,t) = \mathit{Vac}$; assume now that it is. Then let $t' \in J$ be the starting time in χ of the work period of C containing t, and let $\eta^*(x,t)$ be the value decoded from $\eta(C,t')$. In more detail, as said at the end of Section 5.2, we apply the decoding ψ^* to the interior of the colony it to obtain $\eta(x,t)$. ⌟

Note that this definition has the property (C1) of cleanness required in Definition 6.2.

11 Isolated bursts

Here, we will prove that the healing procedure indeed deals with isolated bursts. Our goal is to show that the healing procedure provides relief, as required in an admissible annotated trajectory.

Bursts can create dirt. For its elimination we will rely on the Dwell Cleaning, Spill Bound and the Attack Cleaning properties of a trajectory, see Definition 6.7.

Let us first see how the head can escape dirt.

Lemma 11.1 (Local escape) *Let G be an interval of size nB where $n < Z$. Then in the absence of noise, the head will either escape G within time $O(nT)$, or at some point during this time, it will be inside a clean hole, as per Definition 6.6.*

Proof. Let $c = (c_{\mathrm{attack}} + 2c_{\mathrm{spill}})$, as used in the Dwell Cleaning property of Definition 6.7. Let us cover G by consecutive intervals of size cB called *blocks*, let m be the number of these blocks. Assume that the head does not escape G within time $m \cdot c_{\mathrm{dwell}}T$. Then there is a block K in which it spends cumulative time $c_{\mathrm{dwell}}T$, and the Dwell Cleaning property of trajectories implies that at some point during this time, it will be inside a clean hole in K. □

In a clean configuration, whenever healing started with an alarm, the procedure will be brought to its conclusion as long as no new burst occurred. Now, however, the trajectory properties do not allow any conclusion about the state whenever the head emerges from possible dirt. This complicates the reasoning, and may require several restarts of the healing procedure. By design, the healing procedure can change the *cCore* track only in one cell, even if the head emerged from dirt, with wrong information. The following lemma limits even this kind of damage: it says that the head with the wrong information may increase some existing island, but will not create any new one.

Lemma 11.2 *In the absence of noise, no new island will arise.*

Proof. The islands are defined only by the *cCore* track. In normal mode, this track changes only at the front. If this is not the real front, then we are already in or next to an island.

The healing procedure's change of *cCore* is part of a stitching operation. Looking at the different cases of the proof of Lemma 9.5, we see that inside a healthy area, healing can only change the *cCore* track in two ways.

The first way is case 1, when the *cSw* values were changed in a segment not belonging to the interior. Such an operation can be applied to any healthy configuration without affecting health.

The second case is when the front is moved left or right. This does not affect health either. □

The following lemma is central to the analysis of the behavior of the machine under the condition that bursts are isolated.

Lemma 11.3 (Healing) *In the absence of noise in M^*, the history can be super-annotated. Also, the decoded history $(\eta^*, Noise^*)$ satisfies the Transition Function property of trajectories (Definition 6.7).*

Proof. The proof of super-annotation is by induction on time, extending the super-annotation into the future. The extension is straightforward as long as no distress event is encountered. The Transition Function property is observed, as no obstacle arises to the simulation. Stains do not cause problems: the computation stage of the simulation cycle eliminates them using the error-correcting code.
Consider now the occurrence of a distress event. This can be due to either a burst or the encounter with an island. In the latter case, before the relief a burst can still hit. In the analysis below, then we will just cut our losses and restart, knowing that burst cannot hit again before relief.
At the time of the distress event, let us draw an interval I of size QB centered around the head. The head will not leave it before relief, so we will consider the changes of the configuration inside it. Let $S_1, \dots, S_m$ be the list of substantial segments of I, and $A_1, \dots, A_{m-1}$ the ambiguous segments between them. Note that $m = O(1)$. Let R be the set of those i for which $S_i \cup A_i \cup S_{i+1}$ is clean. For $i \in R$ let n_i be the number of stitching steps by the algorithm of Lemma 9.5 needed to stitch them together. Let $N = \sum_{i \in R} n_i$. Whenever the head enters dirt, we will mark the edge where it entered as *attacked*.
We introduce a few variables for the proof.

- $D =$ be the total size of unclean intervals, divided by B.
- E = the number of un-attacked boundaries where the head can exit without entering. This number never increases.

- F = the number of un-attacked boundaries where the head cannot exit without entering.
- P = be the number of steps that the front moves forward (including the ones after possibly changing the meaning of forward at a regular turn).

The following kinds of event may occur:

(h1) With a stitching done, N decreases by 1 while possibly moving the front backward.

(h2) The head enters dirt, decreasing F.

(h3) The head leaves dirt on a non-attacked edge, decreasing E.

(h4) The head leaves dirt on an attacked edge, decreases the dirt by at least B and increases F by 1.

(h5) The head creates a clean hole, as per Definition 6.6. This may increase E by 2, but also decreases the dirt by $\Omega(B)$.

(h6) A set A_i becomes clean: then i gets added to R; this can happen only m times.

(h7) N increases in the healing mode by 1. This can only happen after the head entered the clean area in healing mode (with the wrong information). At the exit, either E or D had to decrease.

(h8) The head approaches the front: if it does not reach there then it gets closer by $\Omega(\beta)B$.

(h9) The front moves forward in normal mode (possibly after making a regular turn, and thus changing the forward direction). This may also increase N by 1 (and is the only way to do so), say by decreasing a segment S_i. But it is followed by zigging. The zigging may hit dirt, leading to case (h2), or find something wrong—and trigger new healing, which leads to some of the other events. Otherwise the island around the front will be deleted, and the head becomes distress-free.

The above possibilities suggest a potential function

$$N + c_D D + c_E E + c_F F + c_m m - c_P P$$

where $c_D, \ldots$ are appropriate positive constants. Let us look at what each possibility does to the potential.

Cases (h1-h3) decrease the potential if $c_P < 1$. Cases (h4-h5) decrease the potential if $c_D > c_F$. There are only $O(1)$ cases of type (h6), increasing N by a total of $O(\beta)$.

In case (h7), if c_D and c_E are large enough, the increase in N can be charged to a decrease in D or E.
Case (h8) will not change the potential but there are at most $O(1)$ consecutive such cases.
Consider case (h9). If the zigging hits new dirt and $c_F > 1$ then the potential decreases. If it triggers new healing then this will lead to one of the other cases, compensating for the increase of N if the constants (other than c_P) are large.
These considerations show that relief indeed follows distress in time $O(\beta^2 T)$. At that point a normal-mode step moving the front has been made, followed by zigging. Therefore the island causing the distress must have been eliminated, allowing the simulation to continue. The stain caused by the island remains, but as discussed after Definition 10.3, the simulation guarantees that the size and number of stains remains bounded as required by that definition. The simulation also continues to deliver the Transition Function property of trajectories. □

12 Cleaning

12.1 Escape

Let us prepare the proof of the scale-up of the trajectory properties.

The following lemma is similar to Lemma 9.6, but deals with longer intervals.

Lemma 12.1 *Let $I = [a, b)$ be a clean interval of size $> 13QB$. If the head passes it without a burst then a subinterval of size $|I| - 10QB$ becomes clean for M^*.*

Proof. Assume, without loss of generality, that the head passes I from left to right. In the absence of noise, and inside a clean area, the healing procedure can only fail if it starts a rebuilding process. And a rebuilding process never fails.

If the head passes I in normal mode, then the simulation makes I clean for M^*. It may encounter a colony that is only healthy, not super healthy; however, then the simulation will turn it into a healthy one. If healing is invoked but no rebuilding then we can follow the proof of Lemma 9.6: the result of each successful healing is consistent with the the continuation of a simulation.

If a rebuilding is invoked whose whole range falls within I then it succeeds. Then simulation continues, with possible healings thrown in, until new rebulding starts. Lemma 9.7 shows that rebuilding will only extend the super-healthy area. □

The following lemma shows that a clean interval cannot hold the head too long.

Lemma 12.2 (Clean escape) *Suppose that G is a clean interval of size nB. Then within $t = O(n + \beta|n - 2Z|^{+})$ steps, in the absence of noise, one of the following cases happens to the head:*

(A) *it leaves G;*

(B) *it is in a complete clean rebuilding area (continuing the work of rebuilding);*

(C) *it is in the interior of a colony, which is at least $0.5QB$ inside a healthy area.*

(D) *it is in the healthy interior of a—possibly partial—colony (which might need cells outside G to complete it), whose sweep number increases by $\Omega(t/\beta Q)$.*

When the head leaves G we will say that it *escapes*. The bound $O(n + |n - 2/Z|^{+})$ is $O(n)$ if $n < 2Z$, but is $O(\beta n)$ otherwise. Case (D) encompasses case (C). The latter is included just for convenient later reference.

Proof. 1. Suppose that at some time while the head is inside G, the rebuild procedure is started, from a base of rebuild-marked cells large enough not to result in alarm (renewed call for healing) after the first zigging. Then case (A) or (B) happens within $O(\beta n)$ steps.

Proof. The rebuilding procedure is outlined in Section 9.3. From a sufficiently large base, it marks a whole rebuilding area (if it stays inside G), resulting in alternative (B). This happens in $O(\beta n)$ steps, with zigs of size $O(\beta)$. Of course, if $n < Z$ then the head escapes in n steps.

2. Assume that at the beginning, the mode is normal. Then within $O(\beta n)$ steps either a healing call happens, or we have case (A), (C) or (D).

Proof. The head begins or continues a sweeping motion that, in case it does not escape and sees no incoordination to call for healing, passes (with zigging) over the healthy interior of a colony, possibly several times. This leads to case (D).

3. Suppose that at some time t, at position x, while the head is inside G, healing will be called. Then one of the cases of the lemma occur in $O(\beta n)$ steps.

   Proof. If healing fails then it creates a base for rebuilding and calls the rebuilding procedure, leading to the case of part 1 above. *In the discussion below, we will not mention again the possibility that the head leaves G or that a healing ends unsuccessfully: the result in these cases is considered known.*

   If healing is repeated, every invocation of it will bring some progress, described in the proof of Lemma 11.3. Eventually, it will succeed, and the front will continue performing the regular sweeping motion of the simulation. New inconsistencies may start new healing; if this succeeds then after it only the same old simulation can continue. (Lemmas 8.8 and 9.6 imply that the overlapping successful healing areas can be combined.) This leads to case (D).

4. Let us complete the proof.

   Consider all the possibilities for the state at the beginning. We will again leave it unmentioned that the head can always escape (case (A)). The case when we start from normal mode is discussed in part 2. If the head is in healing mode then within one call of the healing procedure, we end up in one of the other cases. Suppose now that the head is in rebuilding mode. If alarm is not called within $O(\beta n)$ steps then the head either builds sweeps a whole rebuilding area, taking us to case (B), or starts or continues erasing the rebuild marks, ending eventually in either alarm or normal mode.

□

We will apply Lemma 12.2 together with the basic trajectory properties in Definition 6.7 (in particular the definition of the constant c_{attack} there).

Lemma 12.3 (Escape) *Suppose the head is in some interval G in some configuration, at some time. In the absence of noise, then within time $O(\beta n^2 T)$, one of the cases* (A)-(C) *of Lemma 12.2 occurs.*

In difference to Lemma 12.2, we do not assume that the interval G is clean. On the other hand, note that the upper bound increased from $O(\beta n T)$ to $O(\beta n^2 T)$. The escape can indeed take longer in our proof.

Imagine that $n/2$ cells in the middle of G are clean representing the middle part of a colony, and the rest is not. The head may perform regular simulation over these $n/2$ cells. As it leaves the clean area on either side, it returns almost immediately, in a state allowing it to sweep again. The only effect is an increase of the clean area by B, so it may take $\Omega(n)$ sweeps to escape.

Proof. Lemma 12.2 asserts an upper bound $O(\beta n)$ on the number of steps needed to either reach cases (A)-(C) or increase the sweep number in case (A) of that lemma, in a clean interval intersecting G. Let $c_{\text{clean-esc}}$ be a constant such that this is $\le c_{\text{clean-esc}}\beta n$ steps. Recall clean holes of Definition 6.6, which we will just call *holes* here. We will show that while the head does not escape G, progress will be made every $O(n)$ steps: holes keep arising and growing in G. The head may spend considerable time inside a hole, but we will limit this time by showing that during it also a certain kind of progress will be made.

1. Assume that a right attacking event of the Attack Cleaning property of a trajectory takes place at point x (thus the interval $[x - c_{\text{attack}}B, x + B)$ is clean). After this event, until the head returns to the left of $x - c_{\text{attack}}B$, we will call the point $x - c_{\text{spill}}B$ an *attacked boundary*. If a hole does not contain the head then its edge closer to the head is an attacked boundary (except the first time the head passed there). By the Spill property before the return of the head the right edge of the clean interval may shrink to $x - c_{\text{spill}}B$, but this is temporary: after the return, it extends to at least $x + 2B$.
2. Let $f = \max(c_{\text{clean-esc}}\beta, c_{\text{dwell}})$. During the time interval considered, in any time subinterval I of size fnT, one of the following kinds of progress will be made:

   (p1) One of the cases (A-C) of Lemma 12.2 occurs.

   (p2) The head enters a clean hole (possibly resulting in the merge of two clean holes).

   (p3) The head leaves a clean hole.

   (p4) A new clean hole appears.

   (p5) The sweep number increases in some clean area intersecting G, as in case (D) of Lemma 12.2.

   Proof. Suppose that the head is in a hole of size kB: let us apply Lemma 12.2. Within time $c_{\text{clean-esc}}k\beta T$ either case (A) of that lemma occurs, leading to our case (p3), or cases (B,C) in which case we are done, or case (D), which is our case (p5).

Assume that G is partitioned into *blocks*: subintervals of size $1.5c_{\text{attack}}B$. Suppose that the head is in an interval L between holes, which it does not leave for the time interval I considered. Let L intersect with k blocks. The head spends on average a cumulative time $\geq fnT/k \geq fT$ over these blocks, so on one of them it spends at least this much time. By the trajectory property called Dwell Cleaning, if $f \geq c_{\text{dwell}}$ then at some time during I, the head will find itself in a clean hole: in other words, either case (p2) or case (p4) occurs.

3. The total number of events of type (p5) is $O(n)$.

   Proof. Each such event is associated with some colony C contained in some hole H. We claim that each colony C is involved in only $O(Q)$ of these events, and the interiors of all these colonies are disjoint, therefore their number is $O(n/Q)$. This limits the number of these events to $O(n)$.

   Without loss of generality, assume that C is on the right edge of H. The total number of these events is $O(Q)$ (even if during different intervals after the head leaving and entering H). Indeed, in $O(Q)$ sweeps, a whole colony simulation cycle is completed. At its end, it is followed by transfer, either to the right or to the left. If it is to the right then there is no returning of the head without moving the edge by at least QB, and thus paying for all the $O(Q)$ sweeps. Transfering left can occur at most twice, because of the feathering property of the simulated machine.

   The interiors of two such colonies C_1, C_2 are necessarily disjoint. Indeed, suppose that an event of type (p5) occurred first in colony C_1. If C_2 intersects C_1 (without being equal to it) then a total restructuring of C_1 must have happened, but this must have involved the rebuilding procedure, moving many times over an interval much wider than $C_1 \cup C_2$, widening the hole—therefore C_2 could not appear on the edge of a hole.

4. Let us finish the proof.

   Each clean hole can be entered at most once on a non-attacked boundary. When it is entered on an attacked boundary, the hole increases at least by B. It follows that except for the steps when the hole is entered the first time, or when a hole is created that intersects the boundary of G, the possibilities (p3-p4) increase the clean part of G by B. So there are only $O(n)$ events of type (p2-p4). By part 3 there are also at most $O(n)$

events of type (p5).

□

12.2 Attack cleaning

Eventually, we will prove the Attack Cleaning property of trajectories (Definition 6.7) for the decoded history $(\eta^*, \mathit{Noise}^*)$. However, first we prove a weaker version, requiring the space-time rectangle of interest to be free not only of Noise^* but also of Noise—that is burst-free.

Lemma 12.4 *Consider a trajectory η in a noise-free space-time rectangle. Here, the decoded history η^* satisfies the Attack Cleaning property.*

Proof. The property says the following for the present case. For current cell x, suppose that the interval $[x - c_{\text{attack}}QB, x + QB)$ is clean for M^*. Suppose further that the transition function, applied to $\eta^*(x, t)$, directs the head right. Then by the time the head comes back to $x - c_{\text{attack}}QB$, the right end of the interval clean in M^* containing x advances to the right by at least QB.

The computation phase of the simulation on the colony of x is completed without the disturbing effect of noise: even the zigging does not go beyond the boundary. Then the transfer phase begins which enters the unclean area to the right of $x + QB$.

We argue that there are only two ways for the head to get back to $x - c_{\text{attack}}QB$.

(a1) The transfer into a new colony with starting point $y \geq x + QB$ succeeds despite the uncleanness, and the clean interval extends over it, before the head moves left to $x - c_{\text{attack}}QB$ in the course of the regular simulation. Some inconsistencies may be discovered along the way, but they are corrected by healing.

(a2) The inconsistencies encountered along the way trigger some rebuilding processes. Eventually, a complete, clean rebuilding area is created, the rebuilding succeeds, leaving a clean colony also to the right of $x + QB$.

The Spill Bound property guarantees that the area to the left of $z = x + (Q - c_{\text{spill}})B$ remains clean, therefore the only way for the head to move left of z is by the rules. Suppose that rebuilding is not initiated (and maintained): then moving left can only happen by the normal course of simulation: the transfer stage of the simulation must be carried out, and this requires at least as many attacks to the right as the number of sweeps in the transfer stage. Every attack (followed by return) extends the clean interval further, until the whole target colony becomes clean, and the transfer completed. This is the case (a1).

Recall the definition of the rebuilding procedure in Section 9.3: the rebuilding area extends $2.5Q$ cells to the left and right from its initiating cell. This may become as large as $5QB$ to the left and right. If rebuilding is initiated, its starting position is necessarily to the right of $x + (Q-2)B$: since inconsistency with the colony of x would be discovered already in the last cell of the colony of x.. It then may extend to the left to at most $x + QB - 5QB = x - 4QB$ (this is overcounting, since the cells of the colony of x are all adjacent). Its many sweeps will result in attacks that clean an area to the right of the starting point. The procedure may be restarted several times, but those restartings will also be initiated to the right of $x + (Q-2)B$. Therefore the rebuilding area does not advance beyond $x - 4QB$: if the head moves to the left of this, then the rebuilding must have succeeded. The rebuilding also must find or create a colony manifestly to the right of the restarting site: this will be to the right of $x + QB$, moving this way the boundary of the area clean in M^* by at least QB. □

Note that in the process described in the above proof, it is possible that the rebuilding finds a competing colony C starting at some $y \in x + QB + [-\beta B, 0)$ which slightly (by the size of an island) overlaps from the right with the colony of x. The rebuilding may decide to keep C and to overwrite the rest of the colony of x as a bridge. This does not affect the result.

12.3 Extended cleaning

Let us draw some consequences of the Pass Cleaning property of trajectories (Definition 6.7). We will extend this property to longer intervals, allowing also some bursts. For the following lemma, let us use the following notation for convenience:

$$\beta' = \beta + 2c_{\text{spill}}.$$

Also, for any interval $I = [a, b)$ let $I' = [a + c_{\text{spill}}B, b - c_{\text{spill}}B)$.

Lemma 12.5 *Assume that the head passes $n > 2\pi$ times over and interval I of size $|I| \geq 2\pi B$ in a burst-free way during some time interval J. There* *could be some other times in J when the head is subject to a burst inside I. Assume that the total number of these bursts is $< n/4\beta'$. (We don't count the times when the head enters and leaves I without passing over and without any burst.) Then there is some time during J when I' becomes clean.*

Proof. We introduce a *virtual* time counting. We only count the times when the head passes I. Let this virtual time interval be denoted by K: we can assume it starts at 0. Let $K_1 = (0, \pi]$ and $K_2 = (\pi, n]$, so $K = K_1 \cup K_2$.

For each burst (of size $\le \beta B$), let us extend it left and right by $c_{\text{spill}}B$ to a size $\le \beta' B$. Let D_1 be the union of all those intervals coming from bursts occurring before virtual time π. By the end of time interval K_1 the set $I \setminus D_1$ becomes clean, due to the Pass Cleaning property. In what follows we are looking at how D_1 shrinks during K_2—while some new bursts may delay the shrinking.

Consider intervals $[a, b)$, U such that $[a, b) \subset U'$, and $U \setminus [a, b)$ is clean at virtual time $u > 1$. If no burst occurs at the virtual times $u, u + 1$ (that is during two burst-free passes) then the Attack Cleaning property implies that by the virtual time $u + 2$, already the interval $U' \setminus [a + B, v - B)$ will also be clean. So let us mount a triangle $T \subset I \times K$ over $[a, b)$ which at time $u + 2j$ covers the interval

$$[a + jB, v - jB)\,,$$

and thus its tip is at virtual time $v = u + (b - a)/B$. We will call $v - u$ the *height* of T, denoted by $|T|$. If no burst occurs until virtual time v, then U' becomes clean by virtual time v. In the special case when $[a, b)$ reaches, say, to the right end of the interval I, we create a triangle (with the same slopes) twice as large, whose tip is at the right end of I as well.

Let us mount a triangle of the above type over each interval of the set D_1 at virtual time π, creating a set $\mathcal{T}_0$ of disjoint triangles. While no burst occurs, the Attack Cleaning property confines dirt to these triangles. On the other hand, every burst may create a dirt interval $[x, x + \beta B)$, at some virtual time u. Let us mount a triangle then over $[x - c_{\text{spill}}B, x + (\beta + c_{\text{spill}})B) \times \{u\}$, and add all these triangles to the set $\mathcal{T}_0$ to get a set of triangles $\mathcal{T}$. These are not necessarily disjoint anymore.

If two virtual triangles T_1, T_2 intersect, and T is the smallest virtual triangle containing both, then it is easy to see that $|T| \le |T_1| + |T_2|$. Denote $T = T_1 + T_2$.

Let us now perform the following operation that will create a disjoint set of triangles. We start with $\mathcal{T}$ and if we find two intersecting triangles in it, then we replace them with their sum. Repeat until the remaining set $\mathcal{T}'$ consists of disjoint triangles. By the above remark $\sum_{T\in\mathcal{T}} |T| = \sum_{T\in\mathcal{T}'} |T|$. By the Attack Cleaning property, the complement of these triangles is clean. Let us ignore the triangles on the boundary for a moment. The sum of the heights of the triangles is at most β' times the number of bursts, and thus at most $n/4$. Taking the boundary triangles into account can increase this by at most a factor of 2, to $n/2$. Therefore in the interval K of length $n - \pi > n/2$ there will be virtual times not intersecting with any element of $\mathcal{T}'$. At the corresponding real times, the interval I is clean. □

Lemma 12.6 *Consider an interval K of size $3QB$ and a time interval J in which no burst of M^* occurs. If at least $4\beta'\pi$ bursts occur in K during J then at some time in J the interval K becomes clean in M^*.*

Proof. For $d = 8\beta'$, both positive and negative i, and $j = 0, \dots, d-1$, let

$$K_{ij} = x + 3QB\,[di + j, di + j + 1)\,,\ \ K_i = \bigcup_{j=0}^{d-1} K_{ij} = x + 3QB\,[di, d(i+1))\,,$$

so $K = K_{00}$. Consider the sweeps corresponding to the $4\beta'\pi$ bursts in K. If a sweep does not exit K on either side then Dwell Cleaning and Attack Cleaning of M^* becomes applicable. Assume this does not happen; then either half of these sweeps exits K on the left, or half of them exits it on the right. Without loss of generality assume that they pass on the right. Then they will have to pass the whole interval K_0 (otherwise again Dwell Cleaning and Attack Cleaning of M^* applies), and thus each interval $K_{0,j}$ gets at least this many passes. So, the intervals $K_{0,j}$, $j > 0$ gets $4\beta'\pi/2$ burst-free passes (since the bursts occurred in K_{00}). Also, none of the K_{0j} becomes clean for M^* during the first half of these passes, since then the Attack Cleaning property would clean K_{00} as well during the remaining passes.

Now, for $i = 1, 2, \dots$ we will show that there is an i' with $|i'| \le i$ such that each $K_{i'j}$ gets at least

$$n_i = 4\beta' \times 2^{i-1}\pi$$

burst-free overpasses during J, and $K_{i'j}$ does not become clean during the first half of these. This clearly must break down somewhere, leading to a contradiction.

We have just proved the case $i = 0$. Suppose that the statement was proved up to some i, we will prove it for $i + 1$. In order for the interval $K_{i'j}$ not to become clean for M^*, by Lemmas 12.5, 12.1 the total number of bursts happening in it must be at least $n_i/4\beta' = 2^{i-1}\pi$. This is true of all j, so the total number of bursts in $K_{i'}$ is at least $d2^{i-1}\pi$. If a sweep does not exit $K_{i'}$ on either side then Dwell Cleaning becomes applicable, so either half of these sweeps exits $K_{i'}$ on the left, or half of them exits it on the right. If they pass on the right then let $(i+1)' = i'+1$, otherwise $(i+1)' = i'-1$. Then they will have to pass the whole interval $K_{(i+1)'}$ (again, otherwise Dwell Cleaning applies), and thus each interval $K_{(i+1)',j}$ gets at least $d2^{i-2}\pi$ passes. Now the inequality (6.4) implies $d > 8\beta' = 2 \cdot 4\beta'$, finishing the proof. $\square$

13 Further analyses

13.1 Trouble with the current plan

The current plan seems not to be working. This plan required the trajectory properties Transition Function, Attack Cleaning, Spill Bound, Dwell Cleaning, Pass Cleaning. But there is a question about the conditions of these properties, for example of Pass Cleaning. If we require it only during a noise-free time interval, then Lemmas 12.5 and 12.6 are not applicable. They may become applicable if we require it in a time interval in which noise is sparse. But then it is not clear how to scale it up. These same two lemmas are needed to scale it up, but it seems difficult to prove their noisy version.

13.2 Cleaning in a small number of passes

What is the real number of passes needed to clean an interval?

Example 13.1 One pass is not sufficient. Consider the following local configuration on an interval I. It is covered with colonies (of level 1, representing cells of level 2). The interval is broken up into subintervals $I_1, I_2, \ldots$, each consisting of γ colonies. (Recall that a sparse set of bursts allows a burst in every γ colonies.) These colonies $C_{k,1}, \dots, C_{k,\gamma}$ represent cells $x_{k,1}, \dots, x_{k,\gamma}$. The leftmost cell $x_{k,1}$ in each interval I_k is Z steps behind the front which is on its right, also in I_k. Colony $C_{k,1}$ pretends that the head is observing $x_{k,1}$, and is in the last stage of moving right. So once the head is in $C_{k,1}$, it does not make any move back, and the simulation of a head moving (in a zigging way) from $x_{k,1}$ to $x_{k,\gamma}$ proceeds. Once the head reaches the right end of $C_{k,\gamma}$, a burst moves it into $C_{k+1,1}$.

This way, the head passes I while leaving the organization at level 2. ⌟

The example suggests that maybe two passes are sufficient: on the way back from the right end, the example configuration will trigger rebuilding at every level.

14 After a large burst

Our goal is to show that the simulation $M \to M^*$ defined in Section 10.2 is indeed a simulation. Section 11 shows this as long as the head operates in an area that is clean for the simulated machineM^* (can be super-annotated), and has no noise for machine M^* (that is its bursts on the level of machine M are isolated). In other words, essentially the Transition Function property of Definition 6.7 of trajectories for the simulated machine M^* has been taken care of.

The new element is the possibility of large areas that cannot be super-annotated: they may not even be clean, even on the level of machine M. The Spill Bound, Attack Cleaning, Dwell Cleaning and Pass Cleaning properties still must be proven.

14.1 Spill bound

One of the most complex analyses of this work is the proof that the simulated machine M^* also obeys the Spill Bound. Let us outline the problem.

We are looking at the boundary z of a large area that is clean for the machine M^*: without loss of generality suppose that this is the right boundary. We will be looking at it in a space-time rectangle $[a, b) \times [u, v)$ that is noise-free in M^*. The interesting case has $a < z < b$ with $z - a, b - z = O(QB)$. The assumption allows occasional bursts of noise of the trajectory η of M, but no two of these bursts must occur in a space-time rectangle of size comparable to the size of a colony work period of M. The Spill Bound for trajectory η keeps the dirt of η within $O(B)$ on the left of z while η is noise-free. If we could assume η noise-free then the heal/rebuild procedures would also keep the dirt of η^* from spilling over by more than $O(B^*) = O(QB)$. However, nothing is assumed about the length of the time interval $[u, v)$. If the head would spend all the time within $O(QB)$ of z then due to the Dwell Cleaning property of trajectories, the area would be cleaned out, again preventing spilling. But the head can slide out far to the right of b fast, since the dirty area on the right of z is arbitrarily large. Then it can come back much later to the left of z, and by a burst (allowed since much time has passed) can deposit an island of dirt there. Repeating this process would produce unlimited spillover, not only of the dirt of η^* but even that of η.

This is where the Pass Cleaning property helps. If the above process is repeated π times, the π passes would clean out an interval on the right of z whose size is of the order of QB, while depositing only π islands of dirt to the left of z. Our construction will have $\pi \ll Q$: more precisely, in our hierarchy of generalized Turing machines we will have $\pi_k = k$, $Q_k = k^2$. And $\ll Q$ bursts will still be handled by healing/rebuilding in η.

Of course this sketch is very crude, but it should help motivate the reasoning that follows.

Bibliography

[1] Ilir Çapuni and Peter Gács. A Turing machine resisting isolated bursts of faults. *Chicago Journal of Theoretical Computer Science*, 2013. See also in arXiv:1203.1335. Extended abstract appeared in SOFSEM 2012. 2, 10.1

[2] Bruno Durand, Andrei Romashchenko, and Alexander Shen. Fixed-point tile sets and their applications. *Journal of Computer and System Sciences*, 78(3):731–764, 2012. 2.2

[3] Peter Gács. Reliable computation with cellular automata. *Journal of Computer System Science*, 32(1):15–78, February 1986. Conference version at STOC' 83. 2.2

[4] Peter Gács. Reliable cellular automata with self-organization. *Journal of Statistical Physics*, 103(1/2):45–267, April 2001. See also arXiv:math/0003117 [math.PR] and the proceedings of STOC '97. 2.2, 2.5

[5] G. L. Kurdyumov. An example of a nonergodic homogenous one-dimensional random medium with positive transition probabilities. *Soviet Mathematical Doklady*, 19(1):211–214, 1978. 2.2